

T E C H N I C A L D E E P D I V E

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.



Copyright © 2019-2020 Randall Maas.
All rights reserved.

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

<u>ANKI VECTOR.....</u>	<u>1</u>
<u>A LOVE LETTER TO THE LITTLE DUDE.....</u>	<u>1</u>
<u>PREFACE</u>	<u>1</u>
1.1. VERSION(S)	1
1.2. CUSTOMIZATION AND PATCHING	1
2. ORGANIZATION OF THIS DOCUMENT.....	1
2.1. ORDER OF DEVELOPMENT	3
<u>CHAPTER 1.....</u>	<u>4</u>
<u>OVERVIEW OF VECTOR.....</u>	<u>4</u>
3. OVERVIEW.....	4
3.1. FEATURES	4
4. PRIVACY AND SECURITY	7
5. COZMO	7
6. ALEXA INTEGRATION	7
<u>PART I.....</u>	<u>9</u>
<u>ELECTRONICS DESIGN.....</u>	<u>9</u>
<u>CHAPTER 2.....</u>	<u>11</u>
<u>ELECTRONICS DESIGN DESCRIPTION</u>	<u>11</u>
7. DESIGN OVERVIEW.....	11
7.1. POWER SOURCE AND DISTRIBUTION TREE.....	14
<u>CHAPTER 3.....</u>	<u>15</u>
<u>HEAD-BOARD ELECTRONICS DESIGN DESCRIPTION.....</u>	<u>15</u>
8. THE HEAD-BOARD (THE MAIN PROCESSOR BOARD)	15
8.1. THE APQ8009 PROCESSOR	16
8.2. SPEAKERS	16

8.3. CAMERA.....	16
8.4. THE LCD	17
8.5. TRIM, CALIBRATION SERIAL NUMBERS AND KEYS.....	17
8.6. MANUFACTURING TEST CONNECTOR/INTERFACE.....	17
9. REFERENCES & RESOURCES.....	18

CHAPTER 4.....19

BACKPACK & BASE-BOARD ELECTRONICS DESIGN DESCRIPTION.....19

10. THE BACKPACK BOARD.....	19
10.1. BACKPACK CONNECTION	20
10.2. OPERATION	20
11. THE BASE-BOARD.....	22
11.1. POWER MANAGEMENT.....	23
11.2. ELECTRO-STATIC DISCHARGE (ESD) PROTECTION	26
11.3. STM32F030 MICROCONTROLLER.....	26
11.4. SENSING	27
11.5. MOTOR DRIVER AND CONTROL	28
11.6. COMMUNICATION.....	28
11.7. CONNECTORS & TEST POINTS	29
12. REFERENCES & RESOURCES	31

CHAPTER 5.....32

ACCESSORY ELECTRONICS DESIGN DESCRIPTION.....32

13. CHARGING STATION	32
14. CUBE	33
14.1. OVER THE AIR FIELD UPDATES	34
14.2. REFERENCES & RESOURCES	34

PART II.....35

BASIC OPERATION35

CHAPTER 6.....37

ARCHITECTURE37

15. OVERVIEW OF VECTOR'S COMMUNICATION INFRASTRUCTURE	37
15.1. APPLICATION SERVICES ARCHITECTURE	38
15.2. EMOTION MODEL, BEHAVIOUR ENGINE, ACTIONS AND ANIMATION ENGINE	39
16. STORAGE SYSTEM	39

16.1. ELECTRONIC MEDICAL RECORDS (EMR)	40
16.2. OEM PARTITION FOR OWNER CERTIFICATES AND LOGS	40
17. SECURITY AND PRIVACY	40
17.1. ENCRYPTED FILESYSTEM.....	41
17.2. THE OPERATING SYSTEM	41
17.3. AUTHENTICATION	42
18. CONFIGURATION AND ASSET FILES	42
18.1. CONFIGURATION FILES	42
19. SOFTWARE-HARDWARE LAYERS	44
19.1. THE BASE BOARD INPUT/OUTPUT	44
19.2. THE LCD DISPLAY.....	44
19.3. THE CAMERA	45
20. REFERENCES & RESOURCES	45
 CHAPTER 7.....	 47
 STARTUP	 47
21. STARTUP	47
21.1. QUALCOMM'S PRIMARY AND SECONDARY BOOTLOADER	47
21.2. ANDROID BOOTLOADER (ABOOT)	48
21.3. REGULAR SYSTEM BOOT.....	49
21.4. ABNORMAL SYSTEM BOOT	50
22. SHUTDOWN	50
22.1. TURNING VECTOR OFF (INTENTIONALLY)	50
22.2. UNINTENTIONALLY	51
22.3. GOING INTO AN OFF STATE	51
23. REFERENCES & RESOURCES	51
 CHAPTER 8.....	 53
 POWER MANAGEMENT.....	 53
24. POWER MANAGEMENT	53
24.1. BATTERY MANAGEMENT	53
24.2. RESPONSES, SHEDDING LOAD / POWER SAVING EFFORTS	54
25. CHARGING	55
 CHAPTER 9.....	 57
 BASIC INPUTS AND OUTPUTS	 57
26. BUTTON, TOUCH AND CLIFF SENSOR INPUT.....	57
27. BACKPACK LIGHTS CONTROL	58

CHAPTER 10.....	59
AUDIO INPUT.....	59
28. AUDIO INPUT.....	59
28.1. THE MICROPHONES	60
28.2. SPATIAL AUDIO PROCESSING	60
28.3. NOISE REDUCTION.....	61
28.4. VOICE ACTIVITY DETECTOR AND WAKE WORD.....	61
28.5. CLOUD SPEECH RECOGNITION	62
28.6. CONNECTIONS WITH VIC-GATEWAY AND SDK ACCESS	63
29. REFERENCES AND RESOURCES	64
CHAPTER 11.....	65
MOTION SENSING.....	65
30. MOTION SENSING	65
30.1. ACCELEROMETER AND GYROSCOPE	65
30.2. TILTED HEAD	66
30.3. SENSING MOTION.....	66
30.4. SENSING INTERACTIONS.....	66
31. REFERENCES AND RESOURCES	66
PART III.....	67
COMMUNICATION	67
CHAPTER 12.....	69
COMMUNICATION	69
32. OVERVIEW OF VECTOR'S COMMUNICATION INFRASTRUCTURE	69
33. INTERNAL COMMUNICATION WITH PERIPHERALS	70
33.1. COMMUNICATION WITH THE BASE-BOARD	70
33.2. SERIAL BOOT CONSOLE	71
33.3. USB	71
34. BLUETOOTH LE	71
35. WIFI.....	72
35.1. FIREWALL.....	73
35.2. WIFI CONFIGURATION	74
36. COMMUNICATING WITH MOBILE APP AND SDK.....	74
36.1. CERTIFICATE BASED AUTHENTICATION	74
37. REFERENCES & RESOURCES	76

CHAPTER 13.....77**BLUETOOTH LE COMMUNICATION PROTOCOL.....77**

38. COMMUNICATION PROTOCOL OVERVIEW	77
38.1. SETTING UP THE COMMUNICATION CHANNEL	79
38.2. FRAGMENTATION AND REASSEMBLY	80
38.3. ENCRYPTION SUPPORT	81
38.4. THE RTS LAYER	82
38.5. FETCHING A LOG	83
39. MESSAGE FORMATS	84
39.1. APPLICATION CONNECTION ID.....	85
39.2. CANCEL PAIRING	86
39.3. CHALLENGE	87
39.4. CHALLENGE SUCCESS.....	88
39.5. CLOUD SESSION	89
39.6. CONNECT.....	91
39.7. DISCONNECT.....	92
39.8. FILE DOWNLOAD	93
39.9. LOG.....	94
39.10. NONCE	95
39.11. OTA UPDATE.....	96
39.12. RESPONSE	97
39.13. SDK PROXY.....	98
39.14. SSH.....	99
39.15. STATUS	100
39.16. WIFI ACCESS POINT.....	101
39.17. WIFI CONNECT	102
39.18. WIFI FORGET	103
39.19. WIFI IP ADDRESS	104
39.20. WIFI SCAN	105

CHAPTER 14.....106**THE HTTPS BASED API106**

40. OVERVIEW OF THE SDK HTTPS API	106
40.1. SDK MESSAGE GROUPINGS	106
41. COMMON ELEMENTS.....	108
41.1. ENUMERATIONS	108
41.2. STRUCTURES.....	110
42. ACCESSORIES AND CUSTOM OBJECTS	111
42.1. ENUMERATIONS	111
42.2. EVENTS	114
42.3. CREATE FIXED CUSTOM OBJECT.....	118
42.4. DEFINE CUSTOM OBJECT	119

42.5. DELETE CUSTOM OBJECTS.....	122
43. ACTIONS AND BEHAVIOUR	123
43.1. ENUMERATIONS	123
43.2. EVENTS	124
43.3. CANCEL ACTION BY ID TAG	124
43.4. LOOK AROUND IN PLACE	125
44. ALEXA.....	126
44.1. ENUMERATIONS	126
44.2. EVENTS	126
44.3. ALEXA AUTH STATE.....	127
44.4. ALEXA OPT IN	127
45. ATTENTION TRANSFER	128
45.1. EVENTS	128
45.2. GET LATEST ATTENTION TRANSFER.....	128
46. AUDIO	129
46.1. ENUMERATIONS	129
46.2. EVENTS	130
46.3. APP INTENT.....	131
46.4. MASTER VOLUME.....	132
46.5. SAY TEXT	133
47. BATTERY.....	134
47.1. ENUMERATIONS	134
47.2. BATTERY STATE	134
48. CONNECTION	135
48.1. ENUMERATIONS	135
48.2. EVENTS	136
48.3. CHECK CLOUD CONNECTION	137
48.4. EVENT STREAM	138
48.5. PROTOCOL VERSION	139
48.6. SDK INITIALIZATION	140
48.7. USER AUTHENTICATION.....	141
48.8. VERSION STATE.....	142
49. CUBE	143
49.1. ENUMERATIONS	143
49.2. EVENTS	144
49.3. CONNECT CUBE	144
49.4. CUBES AVAILABLE	145
49.5. DISCONNECT CUBE.....	145
49.6. DOCK WITH CUBE.....	146
49.7. FLASH CUBE LIGHTS.....	147
49.8. FORGET PREFERRED CUBE.....	147
49.9. PICKUP OBJECT	148
49.10. PLACE OBJECT ON GROUND HERE	149
49.11. POP A WHEELIE.....	150
49.12. ROLL BLOCK	151
49.13. ROLL OBJECT	152
49.14. SET CUBE LIGHTS	153
49.15. SET PREFERRED CUBE.....	154

50.	DISPLAY	155
50.1.	EVENTS	155
50.2.	DISPLAY IMAGE RGB	155
50.3.	ENABLE MIRROR MODE	156
50.4.	SET EYE COLOR	156
51.	FACES	157
51.1.	ENUMERATIONS	157
51.2.	STRUCTURES	158
51.3.	EVENTS	158
51.4.	CANCEL FACE ENROLLMENT	160
51.5.	ENABLE FACE DETECTION	161
51.6.	ERASE ALL ENROLLED FACES	162
51.7.	ERASE ENROLLED FACE BY ID	162
51.8.	FIND FACES	163
51.9.	REQUEST ENROLLED NAMES	164
51.10.	SET FACE TO ENROLL	165
51.11.	UPDATE ENROLLED FACE BY ID	166
52.	FEATURES & ENTITLEMENTS	167
52.1.	ENUMERATIONS	167
52.2.	GET FEATURE FLAG	167
52.3.	GET FEATURE FLAG LIST	168
52.4.	UPDATE USER ENTITLEMENTS	169
53.	IMAGE PROCESSING	170
53.1.	ENUMERATIONS	170
53.2.	EVENTS	170
53.3.	CAMERA FEED	171
53.4.	CAPTURE SINGLE IMAGE	172
53.5.	ENABLE IMAGE STREAMING	172
53.6.	ENABLE MARKER DETECTION	173
53.7.	ENABLE MOTION DETECTION	174
53.8.	IS IMAGE STREAMING ENABLED	174
54.	INTERACTIONS WITH OBJECTS	175
54.1.	STRUCTURES	175
54.2.	DRIVE OFF CHARGER	176
54.3.	DRIVE ON CHARGER	176
54.4.	GO TO OBJECT	177
54.5.	TURN TOWARDS FACE	178
55.	JDOCS	179
55.1.	ENUMERATIONS	179
55.2.	STRUCTURES	179
55.3.	EVENTS	180
55.4.	PULL JDOCS	180
56.	MISC, ACCESSORIES AND CUSTOM OBJECTS	181
56.1.	UPLOAD DEBUG LOGS	181
57.	MOTION CONTROL	182
57.1.	DRIVE STRAIGHT	182
57.2.	DRIVE WHEELS	183
57.3.	GO TO POSE	184

57.4. MOVE HEAD.....	185
57.5. MOVE LIFT.....	185
57.6. SET HEAD ANGLE.....	186
57.7. SET LIFT HEIGHT	187
57.8. STOP ALL MOTORS	188
57.9. TURN IN PLACE	189
58. MOTION SENSING AND ROBOT STATE.....	190
58.1. STRUCTURES.....	190
58.2. EVENTS	192
59. ONBOARDING	193
59.1. ENUMERATIONS	193
59.2. EVENTS	194
59.3. ONBOARDING INPUT	195
59.4. ONBOARDING STATE	198
59.5. ONBOARDING COMPLETE REQUEST.....	198
59.6. ONBOARDING WAKE UP REQUEST.....	198
59.7. ONBOARDING WAKE UP STARTED REQUEST.....	199
60. PHOTOS	200
60.1. STRUCTURES.....	200
60.2. EVENTS	200
60.3. DELETE PHOTO.....	201
60.4. PHOTO.....	201
60.5. PHOTOS INFO	202
60.6. THUMBNAIL	203
61. SETTINGS AND PREFERENCES	204
61.1. STRUCTURES.....	204
61.2. UPDATE SETTINGS.....	204
61.3. UPDATE ACCOUNT SETTINGS	205
62. SOFTWARE UPDATES	206
62.1. ENUMERATIONS	206
62.2. START UPDATE ENGINE	206
62.3. CHECK UPDATE STATUS.....	207
62.4. UPDATE AND RESTART	207

CHAPTER 15..........208

THE CLOUD SERVICES208

63. CONFIGURATION	208
64. JDOCS SERVER	208
64.1. JDOCS INTERACTION.....	209
64.2. DELETE DOCUMENT	209
64.3. ECHO TEST	209
64.4. READ DOCUMENTS	210
64.5. READ DOCUMENT ITEM	210
64.6. WRITE DOCUMENT.....	210
65. NATURAL LANGUAGE PROCESSING.....	210

66. LOG UPLOADER	211
67. CRASH UPLOADER	211
68. DAS MANAGER	211
69. REFERENCES AND RESOURCES	212

PART IV	213
----------------------	------------

ADVANCED FUNCTIONS	213
---------------------------------	------------

CHAPTER 16.....	215
------------------------	------------

IMAGE PROCESSING.....	215
------------------------------	------------

70. CAMERA OPERATION	215
70.1. CAMERA CALIBRATION	216
70.2. VISION MODES	217
70.3. ILLUMINATION LEVEL SENSING	218
71. THE CAMERA POSE: WHAT DIRECTION IS CAMERA POINTING IN?.....	219
72. MARKERS	220
72.1. THE INITIAL PREPARATION STEPS.....	221
72.2. DETECT AND ANALYZE SQUARES	221
72.3. DECODING THE SQUARES	222
72.4. REVAMPING SIZE AND ORIENTATION.....	222
72.5. INFERRING KNOWLEDGE ABOUT OBJECTS	222
73. FACE AND FACIAL FEATURES RECOGNITION.....	223
73.1. FACE DETECTION.....	223
73.2. FACE IDENTIFICATION AND TRAINING	224
73.3. COMMUNICATION INTERFACE	224
74. TENSORFLOW LITE, DETECTING HANDS, PETS... AND THINGS?	225
75. PHOTOS/PICTURES	226
75.1. COMMUNICATION INTERFACE	226
76. RESOURCES & RESOURCES	227

CHAPTER 17.....	229
------------------------	------------

MAPPING & NAVIGATION	229
---------------------------------------	------------

77. MAPPING OVERVIEW	229
78. MAP REPRESENTATION	229
78.1. QUAD-TREE MAP REPRESENTATION BASICS.....	230
78.2. THE MAP'S STARTING POINT.....	230
78.3. HOW THE MAP IS SENT FROM VECTOR TO SDK APPLICATIONS.....	231
79. BUILDING THE MAP	231
79.1. MAPPING CLIFFS AND EDGES.....	231
79.2. TRACKING OBJECTS.....	231

79.3. TIME OF FLIGHT.....	232
79.4. BUILDING A MAP WITH SLAM	232
80. NAVIGATION AND PLANNING.....	232
81. RESOURCES & RESOURCES	233

CHAPTER 18.....234

ACCESSORIES.....234

82. ACCESSORIES IN GENERAL.....	234
82.1. DOCKING	234
83. HOME & CHARGING STATION	234
83.1. DOCKING	234
84. COMPANION CUBE.....	235
84.1. COMMUNICATION.....	235
84.2. CUBE ANIMATIONS	236
85. CUSTOM ITEMS	237

PART V239

ANIMATION.....239

CHAPTER 19.....241

ANIMATION.....241

86. ANIMATION TRIGGERS AND ANIMATIONS	241
86.1. ANIMATION TRACKS	241
87. ANIMATION FILES	242
88. SDK COMMANDS TO PLAY ANIMATIONS	242

CHAPTER 20.....244

VIDEO DISPLAY & FACE244

89. OVERVIEW OF THE DISPLAY	244
89.1. ORIGIN.....	244
90. RENDERING SYSTEM	245
90.1. FULL-SCREEN SPRITES	245
90.2. COMPOSITION SYSTEM	246
91. PROCEDURAL FACE	246
91.1. THE RENDERING OF INDIVIDUAL EYES	247
91.2. THE PROCESS OF DRAWING THE PROCEDURAL FACE	248
92. COMMANDS	248

CHAPTER 21.....	249
AUDIO OUT.....	249
93. SPEAKER.....	249
93.1. SOUND FILES.....	249
93.2. VOLUME CONTROL.....	249
93.3. TEXT TO SPEECH.....	250
94. PARAMETERIC SOUND EFFECTS.....	252
95. COMMANDS.....	252
96. REFERENCES AND RESOURCES	252
CHAPTER 22.....	253
MOTION CONTROL	253
97. MOTION CONTROL	253
97.1. FEEDBACK	254
97.2. MOTOR CONTROL	254
97.3. DIFFERENTIAL DRIVE KINEMATICS	254
98. MOTION CONTROL COMMANDS.....	255
CHAPTER 23.....	256
ANIMATION FILE FORMAT.....	256
99. ANIMATION BINARY FILE FORMAT	256
99.1. OVERVIEW OF THE FILE FORMAT	256
99.2. RELATIONSHIP WITH COZMO	257
100. STRUCTURES.....	257
100.1. ANIMCLIPS.....	257
100.2. ANIMCLIP	257
100.3. AUDIOEVENTGROUP	257
100.4. AUDIOPARAMETER	258
100.5. AUDIOSTATE	258
100.6. AUDIOSWITCH	258
100.7. BACKPACKLIGHTS.....	258
100.8. BODYMOTION	259
100.9. EVENT	259
100.10. FACEANIMATION	259
100.11. HEADANGLE	259
100.12. LIFTHEIGHT	260
100.13. KEYFRAMES.....	260
100.14. PROCEDURALFACE.....	261
100.15. RECORDHEADING.....	262

100.16. ROBOTAUDIO	262
100.17. TURNTORECORDEDHEADING	262
PART VI	263
HIGH LEVEL AI.....	263
CHAPTER 24.....	265
BEHAVIOR	265
101. OVERVIEW	265
102. ACTIONS AND BEHAVIORS.....	265
102.1. ACTIONS AND THE ACTION QUEUES.....	265
102.2. BEHAVIORS	265
103. EMOTIONS, AND STIMULATION	266
103.1. STIMULATION	266
103.2. THE EMOTION MODEL	266
103.3. MOOD MANAGER.....	267
104. REFERENCES & RESOURCES	269
PART VII	271
MAINTENANCE	271
CHAPTER 25.....	273
SETTINGS, PREFERENCES, FEATURES, AND STATISTICS	273
105. THE ARCHITECTURE	273
106. WIFI CONFIGURATION	274
107. THE OWNER ACCOUNT INFORMATION	274
108. PREFERENCES & ROBOT SETTINGS.....	275
108.1. ENUMERATIONS	275
108.2. ROBOTSETTINGSCONFIG.....	276
109. OWNER ENTITLEMENTS	277
110. VESTIGAL COZMO SETTINGS	278
111. FEATURE FLAGS.....	278
111.1. CONFIGURATION FILE	278
111.2. COMMUNICATION INTERFACE TO THE FEATURES	279
112. ROBOT LIFETIME STATISTICS & EVENTS	279
113. REFERENCES & RESOURCES	279
CHAPTER 26.....	280

THE SOFTWARE UPDATE PROCESS.....	280
114. THE ARCHITECTURE	280
115. THE UPDATE FILE.....	280
115.1. MANIFEST.INI.....	281
115.2. HOW TO DECRYPT THE OTA UPDATE ARCHIVE FILES	282
116. THE UPDATE PROCESS	283
116.1. STATUS DIRECTORY.....	283
116.2. PROCESS.....	283
117. RESOURCES & RESOURCES	284
CHAPTER 28.....	285
DIAGNOSTICS	285
118. OVERVIEW	285
118.1. THE SOFTWARE INVOLVED	285
119. SPECIAL SCREENS AND MODES	287
119.1. CUSTOMER CARE INFORMATION SCREEN.....	287
119.2. VECTORS' DEBUG SCREEN (TO GET INFO FOR USE WITH THE SDK).....	287
119.3. DISPLAY FAULT CODES FOR ABNORMAL SYSTEM SERVICE EXIT / HANG.....	287
119.4. RECOVERY MODE	287
119.5. "FACTORY RESET".....	288
120. BACKPACK LIGHTS	288
121. DIAGNOSTIC COMMANDS	288
122. LOGS	288
122.1. GATHERING LOGS, ON DEMAND.....	289
123. CONSOLE FILTER	289
124. USAGE STUDIES AND PROFILING DATA	291
124.1. EVENT TRACING	291
124.2. PROFIILING AND LIBOSSTATE.....	292
124.3. EXPERIMENTS	293
125. REFERENCES & RESOURCES	294
REFERENCES & RESOURCES	295
126. CREDITS.....	295
127. REFERENCE DOCUMENTATION AND RESOURCES	295
127.1. ANKI.....	295
127.2. OTHER	295
127.3. QUALCOMM	296
APPENDICES	297
APPENDIX A.....	299

ABBREVIATIONS, ACRONYMS, GLOSSARY	299
APPENDIX B	304
TOOL CHAIN	304
128. REFERENCES & RESOURCES	306
APPENDIX C	308
ALEXA MODULES	308
APPENDIX D	310
FAULT AND STATUS CODES	310
APPENDIX E	312
FILE SYSTEM	312
APPENDIX F	317
BLUETOOTH LE SERVICES & CHARACTERISTICS.....	317
129. CUBE SERVICES	317
129.1. CUBE'S ACCELEROMETER SERVICE	318
130. VECTOR SERVICES SERVICE	318
130.1. VECTOR'S SERIAL SERVICE	319
APPENDIX G	320
SERVERS & DATA SCHEMA	320
APPENDIX H	322
FEATURES	322
APPENDIX I	324
PHRASES AND THEIR INTENT	324

APPENDIX J.....	328
<hr/>	
DAS TRACKED EVENTS AND STATISTICS	328
<hr/>	
131. DAS TRACKED EVENTS AND STATISTICS	328
131.1. BASIC INFORMATION	328
131.2. POWER MANAGEMENT EVENTS AND STATISTICS	329
131.3. MOTOR AND IMU STATISTICS AND EVENTS.....	330
131.4. COMMUNICATION RELATED EVENTS POSTED TO DAS	330
131.5. UPDATE-RELATED EVENTS POSTED TO DAS	331
131.6. BEHAVIOUR, FEATURE, MOOD, AND ENGINE RELATED EVENTS POSTED To DAS	332
131.7. ROBOT LIFETIME STATISTICS & EVENTS	332
<hr/>	
APPENDIX K.....	335
<hr/>	
PLEO.....	335
<hr/>	
131.8. SALES.....	336
131.9. RESOURCES	336

Figures

FIGURE 1: VECTOR'S MAIN FEATURES	4
FIGURE 2: VECTOR'S MAIN ELEMENTS.....	11
FIGURE 3: CIRCUIT BOARD TOPOLOGY.....	13
FIGURE 4: VECTOR'S MAIN MICROCONTROLLER CIRCUIT BOARDS.....	13
FIGURE 5: POWER DISTRIBUTION	14
FIGURE 6: HEAD-BOARD BLOCK DIAGRAM	15
FIGURE 7: BACKPACK BOARD BLOCK DIAGRAM.....	19
FIGURE 8: BASE-BOARD BLOCK DIAGRAM	22
FIGURE 9: A REPRESENTATIVE BATTERY CONNECT SWITCH.....	24
FIGURE 10: A REPRESENTATIVE PFET BASED REVERSED POLARITY PROTECTION.....	25
FIGURE 11: CHARGING PROFILE (ADAPTED FROM TEXAS INSTRUMENTS).....	25
FIGURE 12: MOTOR DRIVER H-BRIDGE	28
FIGURE 13: CHARGING STATION MAIN FEATURES	32
FIGURE 14: CHARGING STATION BLOCK DIAGRAM	32
FIGURE 15: CUBE'S MAIN FEATURES.....	33
FIGURE 16: BLOCK DIAGRAM OF THE CUBE'S ELECTRONICS	33
FIGURE 17: THE OVERALL FUNCTIONAL BLOCK DIAGRAM	37
FIGURE 18: THE OVERALL COMMUNICATION INFRASTRUCTURE	38
FIGURE 19: THE BEHAVIOUR-ANIMATION FLOW	39
FIGURE 20: THE BASEBOARD-RELATED ARCHITECTURE	44
FIGURE 21: THE LCD ARCHITECTURE.....	44
FIGURE 22: THE CAMERA ARCHITECTURE.....	45
FIGURE 23: THE BATTERY LEVEL CLASSIFICATION TREE	53
FIGURE 24: THE BATTERY THRESHOLDS	54
FIGURE 25: THE RESPONSE TO BATTERY LEVEL	55
FIGURE 26: THE TOUCH AND BUTTON INPUT ARCHITECTURE.....	57
FIGURE 27: THE BACKPACK LIGHTS OUTPUT ARCHITECTURE.....	58
FIGURE 28: THE AUDIO INPUT FUNCTIONAL BLOCK DIAGRAM	59
FIGURE 29: THE AUDIO INPUT ARCHITECTURE.....	60
FIGURE 30: TYPICAL SPATIAL AUDIO PROCESSING FLOW.....	60
FIGURE 31: TYPICAL AUDIO NOISE REDUCTION FLOW	61
FIGURE 32: THE SPEECH RECOGNITION	62
FIGURE 33: MOTION SENSING.....	65
FIGURE 34: THE OVERALL COMMUNICATION INFRASTRUCTURE	69
FIGURE 35: THE BLUETOOTH LE STACK.....	72
FIGURE 36: THE WiFi STACK.....	73
FIGURE 37: OVERVIEW OF ENCRYPTION AND FRAGMENTATION STACK	78
FIGURE 38: SEQUENCE FOR INITIATING COMMUNICATION WITH VECTOR	79
FIGURE 39: THE FORMAT OF AN RTS FRAME.....	82
FIGURE 40: SEQUENCE FOR INITIATING COMMUNICATION WITH VECTOR	83
FIGURE 41: THE CAMERA ARCHITECTURE.....	215
FIGURE 42: THE CAMERA FIELD OF VIEW	216
FIGURE 43: THE PROCESSING OF THE IMAGE FOR SYMBOLS AND OBJECTS.....	220
FIGURE 44: A TYPICAL RECTANGLE AROUND THE VISUAL MARKERS.....	220
FIGURE 45: PREPARING IMAGE FOR SCANNING FOR SYMBOLS.....	220
FIGURE 46: DECODING THE SYMBOL.....	222
FIGURE 47: THE FACE DETECTION AND RECOGNITION PROCESSES	223
FIGURE 48: THE PROCESSING OF THE IMAGE FOR SYMBOLS AND OBJECTS.....	225
FIGURE 49: VECTOR RECOGNIZING FRUIT DRAWING BY JESSE EASLEY	225
FIGURE 50: TENSORFLOW LITE WITH HARDWARE SPECIFIC ACCELERATORS	225

FIGURE 51: MAPPING CONTEXTS	229
FIGURE 52: STRUCTURE OF A QUAD-TREE	230
FIGURE 53: A TYPICAL LOCALIZATION AND MAPPING FUNCTIONAL BLOCK DIAGRAM	232
FIGURE 54: THE BEHAVIOUR-ANIMATION FLOW	241
FIGURE 55: THE DISPLAY LAYERS.....	245
FIGURE 56: THE DISPLAY ANIMATIONS COMPOSITION	245
FIGURE 57: THE DISPLAY ANIMATIONS FUNCTIONAL FLOW.....	245
FIGURE 58: FACE CONTROL POINTS #1	246
FIGURE 59: BASIC PARAMETERS OF AN INDIVIDUAL EYE CONTROL.....	247
FIGURE 60: PARAMETERS OF AN EYES EYELIDS (OR CHEEKS)	247
FIGURE 61: THE FUNCTIONAL FLOW OF DRAWING THE PROCEDURAL FACE	248
FIGURE 62: THE AUDIO OUTPUT ARCHITECTURE.....	249
FIGURE 63: MOTION CONTROLLER.....	253
FIGURE 64: A TYPICAL MOTOR CONTROLLER	254
FIGURE 65: RADIUS OF ARC MEASUREMENT.....	255
FIGURE 66: THE ANIMATION FILE STRUCTURE.....	256
FIGURE 67: THE ARCHITECTURE FOR STORING PREFERENCES, ACCOUNT INFO, ENTITLEMENTS, AND TRACKING STATS	273
FIGURE 68: THE ARCHITECTURE FOR UPDATING VECTOR'S SOFTWARE	280
FIGURE 69: ALEXA'S FUNCTION BLOCKS (IMAGE COURTESY AMAZON).....	308

Tables

TABLE 1: VECTOR'S MAIN ELEMENTS	12
TABLE 2: VECTOR'S CIRCUIT BOARDS	13
TABLE 3: THE HEAD-BOARDS FUNCTIONAL ELEMENTS	16
TABLE 4: THE CAMERA CONTROLS	17
TABLE 5: BACKPACK BOARD FUNCTIONAL ELEMENTS	20
TABLE 6: THE BASE-BOARD FUNCTIONAL ELEMENTS	22
TABLE 7: HEAD MOTOR CONNECTOR (P1) PIN MAP	29
TABLE 8: LIFT MOTOR CONNECTOR (P2) PIN MAP	29
TABLE 9: FRONT SENSOR (TIME OF FLIGHT) CONNECTOR PIN MAP	30
TABLE 10: DEBUG CONNECTOR PIN MAP	30
TABLE 11: HEAD MOTOR CONNECTOR (P1) PIN MAP	30
TABLE 12: THE CUBE'S ELECTRONIC DESIGN ELEMENTS	33
TABLE 13: VECTOR PROCESSES	38
TABLE 14: ELECTRONIC MEDICAL RECORD (EMR)	40
TABLE 15: OEM PARTITION FILE HIERARCHY	40
TABLE 16: THE PLATFORM CONFIG JSON STRUCTURE	42
TABLE 17: VECTOR SHUTDOWN CODES	50
TABLE 18: BATTERYLEVEL CODES AS THEY APPLY TO VECTOR	54
TABLE 19: THE BACKPACK LEDS JSON STRUCTURE	58
TABLE 20: THE MICTRIGGERCONFIG JSON STRUCTURE	62
TABLE 21: THE TBD JSON STRUCTURE	62
TABLE 22: THE USER_INTENT_MAP JSON STRUCTURE	63
TABLE 23: THE INTENT MAPPING JSON STRUCTURE	63
TABLE 24: JSON STRUCTURE	70
TABLE 25: ELEMENTS OF THE BLUETOOTH LE STACK	72
TABLE 26: ELEMENTS OF THE BLUETOOTH LE STACK	73
TABLE 27: ELEMENTS OF A VECTOR CERTIFICATE	75
TABLE 28: OEM CLOUD FOLDER	75
TABLE 29: CLOUD INFO\${SERIALNUM} STRUCTURE	75
TABLE 30: PARAMETERS FOR HANDSHAKE MESSAGE	79
TABLE 31: THE ENCRYPTION VARIABLES	81
TABLE 32: SUMMARY OF THE COMMANDS	84
TABLE 33: PARAMETERS FOR APPLICATION CONNECTION ID REQUEST	85
TABLE 34: PARAMETERS FOR CHALLENGE REQUEST	87
TABLE 35: PARAMETERS FOR CHALLENGE RESPONSE	87
TABLE 36: PARAMETERS FOR CLOUD SESSION REQUEST	89
TABLE 37: PARAMETERS FOR CLOUD SESSION RESPONSE	89
TABLE 38: CLOUD STATUS ENUMERATION	89
TABLE 39: PARAMETERS FOR CONNECTION REQUEST	91
TABLE 40: PARAMETERS FOR CONNECTION RESPONSE	91
TABLE 41: CONNECTION TYPES ENUMERATION	91
TABLE 42: PARAMETERS FOR FILE DOWNLOAD REQUEST	93
TABLE 43: PARAMETERS FOR LOG REQUEST	94
TABLE 44: LOG FILTER	94
TABLE 45: PARAMETERS FOR LOG RESPONSE	94
TABLE 46: PARAMETERS FOR NONCE REQUEST	95
TABLE 47: PARAMETERS FOR NONCE RESPONSE	95
TABLE 48: PARAMETERS FOR OTA REQUEST	96
TABLE 49: PARAMETERS FOR OTA RESPONSE	96
TABLE 50: OTA STATUS ENUMERATION	96

TABLE 51: PARAMETERS FOR RESPONSE	97
TABLE 52: PARAMETERS FOR THE SDK PROXY REQUEST	98
TABLE 53: PARAMETERS FOR THE SDK PROXY RESPONSE	98
TABLE 54: PARAMETERS FOR SSH REQUEST.....	99
TABLE 55: SSH AUTHORIZATION KEY	99
TABLE 56: PARAMETERS FOR STATUS RESPONSE	100
TABLE 57: WiFi STATE ENUMERATION	100
TABLE 58: PARAMETERS FOR WiFi ACCESS POINT REQUEST.....	101
TABLE 59: PARAMETERS FOR WiFi ACCESS POINT RESPONSE.....	101
TABLE 60: PARAMETERS FOR WiFi CONNECT REQUEST	102
TABLE 61: WiFi AUTHENTICATION TYPES ENUMERATION	102
TABLE 62: PARAMETERS FOR WiFi CONNECT COMMAND.....	102
TABLE 63: PARAMETERS FOR WiFi FORGET REQUEST	103
TABLE 64: PARAMETERS FOR WiFi FORGET RESPONSE.....	103
TABLE 65: PARAMETERS FOR WiFi IP ADDRESS RESPONSE	104
TABLE 66: PARAMETERS FOR WiFi SCAN RESPONSE	105
TABLE 67: PARAMETERS ACCESS POINT STRUCTURE.....	105
TABLE 68: ResultCode ENUMERATION	108
TABLE 69: RobotStatus ENUMERATION	108
TABLE 70: CladPoint JSON STRUCTURE.....	110
TABLE 71: CladRectangle JSON STRUCTURE	110
TABLE 72: PoseStruct JSON STRUCTURE	110
TABLE 73: CustomObjectMarker ENUMERATION.....	111
TABLE 74: CustomType ENUMERATION	112
TABLE 75: ObjectType ENUMERATION	113
TABLE 76: ObjectType ENUMERATION	113
TABLE 77: ObjectEvent JSON STRUCTURE	114
TABLE 78: ObjectAvailable JSON STRUCTURE	114
TABLE 79: ObjectConnectedState JSON STRUCTURE.....	115
TABLE 80: ObjectMoved JSON STRUCTURE	115
TABLE 81: ObjectStoppedMoving JSON STRUCTURE.....	115
TABLE 82: ObjectTapped JSON STRUCTURE	116
TABLE 83: ObjectUpAxis JSON STRUCTURE.....	116
TABLE 84: UpAxis ENUMERATION	116
TABLE 85: RobotObservedObject JSON STRUCTURE	117
TABLE 86: CreateFixedCustomObjectRequest JSON STRUCTURE	118
TABLE 87: CreateFixedCustomObjectResponse JSON STRUCTURE	118
TABLE 88: DefineCustomObjectRequest JSON STRUCTURE	119
TABLE 89: CustomBoxDefinition JSON STRUCTURE	120
TABLE 90: CustomCubeDefinition JSON STRUCTURE	120
TABLE 91: CustomWallDefinition JSON STRUCTURE	121
TABLE 92: DefineCustomObjectResponse JSON STRUCTURE	121
TABLE 93: DeleteCustomObjectsRequest JSON STRUCTURE.....	122
TABLE 94: CustomObjectDeletionMode ENUMERATION.....	122
TABLE 95: DeleteCustomObjectsResponse JSON STRUCTURE	122
TABLE 96: ActionTagConstans ENUMERATION	123
TABLE 97: BehaviorResults ENUMERATION	123
TABLE 98: StimulationInfo JSON STRUCTURE	124
TABLE 99: CancelActionByIdTagRequest JSON STRUCTURE.....	124
TABLE 100: CancelActionByIdTagResponse JSON STRUCTURE	124
TABLE 101: LookaroundInPlaceResponse JSON STRUCTURE	125
TABLE 102: AlexaAuthState ENUMERATION	126
TABLE 103: AlexaAuthEvent JSON STRUCTURE	126

TABLE 104: ALEXAAUTHSTATERESPONSE JSON STRUCTURE.....	127
TABLE 105: ALEXAOPTINREQUEST JSON STRUCTURE	127
TABLE 106: ALEXAOPTINRESPONSE JSON STRUCTURE.....	127
TABLE 107: ATTENTIONTRANSFER JSON STRUCTURE.....	128
TABLE 108: ATTENTIONTRANSFERREASON ENUMERATION.....	128
TABLE 109: GETLATESTATTENTIONTRANSFERRESPONSE JSON STRUCTURE.....	128
TABLE 110: LATESTATTENTIONTRANSFER JSON STRUCTURE.....	128
TABLE 111: MASTERVOLUMELEVEL ENUMERATION.....	129
TABLE 112: UTTERANCESTATE ENUMERATION.....	129
TABLE 113: USERINTENT JSON STRUCTURE.....	130
TABLE 114: WAKEWORD JSON STRUCTURE	130
TABLE 115: WAKEWORDEND JSON STRUCTURE	130
TABLE 116: APPREQUEST JSON STRUCTURE	131
TABLE 117: APPINTENTRESPONSE JSON STRUCTURE.....	131
TABLE 118: MASTERVOLUMEREQUEST JSON STRUCTURE	132
TABLE 119: MASTERVOLUMERESPONSE JSON STRUCTURE	132
TABLE 120: SAYTEXTREQUEST JSON STRUCTURE.....	133
TABLE 121: SAYTEXTRESPONSE JSON STRUCTURE	133
TABLE 122: BATTERYSTATERESPONSE JSON STRUCTURE	134
TABLE 123: CONNECTIONCODE ENUMERATION	135
TABLE 124: CONNECTIONRESPONSE JSON STRUCTURE	136
TABLE 125: EVENT JSON STRUCTURE.....	136
TABLE 126: TIMESTAMPEDSTATUS JSON STRUCTURE.....	137
TABLE 127: STATUS JSON STRUCTURE	137
TABLE 128: CHECKCLOUDRESPONSE JSON STRUCTURE	137
TABLE 129: EVENTREQUEST JSON STRUCTURE	138
TABLE 130: FILTERLIST JSON STRUCTURE	138
TABLE 131: EVENTRESPONSE JSON STRUCTURE.....	138
TABLE 132: PROTOCOLVERSIONREQUEST JSON STRUCTURE	139
TABLE 133: PROTOCOLVERSIONRESPONSE JSON STRUCTURE	139
TABLE 134: RESULT ENUMERATION	139
TABLE 135: SDKINITIALIZATIONREQUEST JSON STRUCTURE.....	140
TABLE 136: SDKINITIALIZATIONRESPONSE JSON STRUCTURE	140
TABLE 137: USERAUTHENTICATIONREQUEST JSON STRUCTURE	141
TABLE 138: USERAUTHENTICATIONRESPONSE JSON STRUCTURE	141
TABLE 139: CODE ENUMERATION	141
TABLE 140: VERSIONSTATERESPONSE JSON STRUCTURE	142
TABLE 141: ALIGNMENTTYPE ENUMERATION.....	143
TABLE 142: CUBEPOWERLEVEL CODES AS THEY APPLY TO VECTOR	143
TABLE 143: CUBEPOWERLEVEL JSON STRUCTURE	144
TABLE 144: CONNECTCUBERESPONSE JSON STRUCTURE	144
TABLE 145: CUBESAVAILABLERESPONSE JSON STRUCTURE	145
TABLE 146: DISCONNECTCUBERESPONSE JSON STRUCTURE	145
TABLE 147: DOCKWITHCUBEREQUEST JSON STRUCTURE	146
TABLE 148: DOCKWITHCUBERESPONSE JSON STRUCTURE.....	146
TABLE 149: FLASHCUBLELIGHTSRESPONSE JSON STRUCTURE.....	147
TABLE 150: FORGETPREFERREDCUBERESPONSE JSON STRUCTURE	147
TABLE 151: PICKUPOBJECTREQUEST JSON STRUCTURE.....	148
TABLE 152: PICKUPOBJECTRESPONSE JSON STRUCTURE	148
TABLE 153: PLACEOBJECTONGROUNDREQUEST JSON STRUCTURE.....	149
TABLE 154: PLACEOBJECTONGROUNDRSPONSE JSON STRUCTURE	149
TABLE 155: POPAWHEELIEREQUEST JSON STRUCTURE	150
TABLE 156: POPAWHEELIERESPONSE JSON STRUCTURE.....	150

TABLE 157: ROLLBLOCKRESPONSE JSON STRUCTURE	151
TABLE 158: ROLLOBJECTREQUEST JSON STRUCTURE.....	152
TABLE 159: ROLLOBJECTRESPONSE JSON STRUCTURE	152
TABLE 160: SETCUBELIGHTSREQUEST JSON STRUCTURE	153
TABLE 161: MAKERELATIVEMODE ENUMERATION.....	154
TABLE 162: SETCUBELIGHTSRESPONSE JSON STRUCTURE	154
TABLE 163: SETPREFERREDCUBEREQUEST JSON STRUCTURE	154
TABLE 164: SETPREFERREDCUBERESPONSE JSON STRUCTURE	154
TABLE 165: DISPLAYFACEIMAGERGBREQUEST JSON STRUCTURE	155
TABLE 166: DISPLAYFACEIMAGERGBRESPONSE JSON STRUCTURE.....	155
TABLE 167: ENABLEREFLACERESPONSE JSON STRUCTURE.....	156
TABLE 168: ENABLEREFLACERESPONSE JSON STRUCTURE	156
TABLE 169: SETEYECOLORREQUEST JSON STRUCTURE.....	156
TABLE 170: SETEYECOLORRESPONSE JSON STRUCTURE	156
TABLE 171: FACIALEXPRESSION ENUMERATION	157
TABLE 172: FACIALEXPRESSION ENUMERATION	158
TABLE 173: ROBOTRENAMEDENROLLEDFACE JSON STRUCTURE.....	158
TABLE 174: FACEENROLLMENTCOMPLETE JSON STRUCTURE	158
TABLE 175: ROBOTCHANGEDOBSERVEDFACEID JSON STRUCTURE	159
TABLE 176: ROBOTOBSERVEDFACE JSON STRUCTURE	159
TABLE 177: CANCELFACEENROLLMENTRESPONSE JSON STRUCTURE	160
TABLE 178: ENABLEFACEDETECTIONREQUEST JSON STRUCTURE.....	161
TABLE 179: ENABLEFACEDETECTIONRESPONSE JSON STRUCTURE	161
TABLE 180: ERASEALLENROLLEDFACESRESPONSE JSON STRUCTURE.....	162
TABLE 181: ERASEENROLLEDFACEBYIDREQUEST JSON STRUCTURE.....	162
TABLE 182: ERASEENROLLEDFACEBYIDRESPONSE JSON STRUCTURE	162
TABLE 183: FINDFACESTRUCTURE	163
TABLE 184: REQUESTENROLLEDNAMESRESPONSE JSON STRUCTURE.....	164
TABLE 185: LOADEDKNOWNFACE JSON STRUCTURE	164
TABLE 186: SETFACETOENROLLREQUEST JSON STRUCTURE	165
TABLE 187: SETFACETOENROLLRESPONSE JSON STRUCTURE.....	165
TABLE 188: UPDATEENROLLEDFACEBYIDREQUEST JSON STRUCTURE	166
TABLE 189: UPDATEENROLLEDFACEBYIDRESPONSE JSON STRUCTURE	166
TABLE 190: USERENTITLEMENT ENUMERATION	167
TABLE 191: FEATUREFLAGREQUEST JSON STRUCTURE.....	167
TABLE 192: FEATUREFLAGRESPONSE JSON STRUCTURE	167
TABLE 193: FEATUREFLAGLISTREQUEST JSON STRUCTURE	168
TABLE 194: FEATUREFLAGLISTRESPONSE JSON STRUCTURE.....	168
TABLE 195: JSON PARAMETERS FOR UPDATEREQUEST	169
TABLE 196: JSON PARAMETERS FOR USERENTITLEMENTS CONFIG	169
TABLE 197: UPDATEREQUEST JSON STRUCTURE.....	169
TABLE 198: IMAGEENCODING ENUMERATION.....	170
TABLE 199: CAMERAFEEDRESPONSE JSON STRUCTURE	171
TABLE 200: CAPTURESINGLEIMAGERESPONSE JSON STRUCTURE	172
TABLE 201: ENABLEREQUEST JSON STRUCTURE	172
TABLE 202: ENABLERESPONSE JSON STRUCTURE	172
TABLE 203: JSON PARAMETERS FOR ENABLEREQUEST	173
TABLE 204: ENABLERESPONSE JSON STRUCTURE	173
TABLE 205: ENABLEREQUEST JSON STRUCTURE	174
TABLE 206: ENABLERESPONSE JSON STRUCTURE	174
TABLE 207: ISIMAGESTREAMINGRESPONSE JSON STRUCTURE	174
TABLE 208: PATHMOTIONPROFILE JSON STRUCTURE	175
TABLE 209: DRIVEOFFCHARGERESPONSE JSON STRUCTURE	176

TABLE 210: DRIVEONCHARGERESPONSE JSON STRUCTURE	176
TABLE 211: GOTOOBJECTREQUEST JSON STRUCTURE	177
TABLE 212: GOTOOBJECTRESPONSE JSON STRUCTURE	177
TABLE 213: TURNTOWARDSFACEREQUEST JSON STRUCTURE	178
TABLE 214: TURNTOWARDSFACERESPONSE JSON STRUCTURE	178
TABLE 215: JDOCTYPE ENUMERATION	179
TABLE 216: JDOC JSON STRUCTURE	179
TABLE 217: JSON NAMEDJDOC STRUCTURE	179
TABLE 218: JSON JDOCSCHANGED REQUEST STRUCTURE	180
TABLE 219: JSON PULLJDOCSREQUEST STRUCTURE	180
TABLE 220: JSON PULLJDOCSRESPONSE STRUCTURE	180
TABLE 221: UPLOADDEBUGLOGSRESPONSE JSON STRUCTURE	181
TABLE 222: DRIVESTRAIGHTREQUEST JSON STRUCTURE.....	182
TABLE 223: DRIVESTRAIGHTRESPONSE JSON STRUCTURE	182
TABLE 224: DRIVEWHEELSREQUEST JSON STRUCTURE	183
TABLE 225: DRIVEWHEELSRESPONSE JSON STRUCTURE.....	183
TABLE 226: GOTOPOSEREQUEST JSON STRUCTURE	184
TABLE 227: GOTOPOSERESPONSE JSON STRUCTURE	184
TABLE 228: MOVEHEADREQUEST JSON STRUCTURE	185
TABLE 229: MOVEHEADRESPONSE JSON STRUCTURE.....	185
TABLE 230: MOVELIFTREQUEST JSON STRUCTURE	185
TABLE 231: MOVELIFTRESPONSE JSON STRUCTURE	185
TABLE 232: SETHEADANGLEREQUEST JSON STRUCTURE	186
TABLE 233: SETHEADANGLERESPONSE JSON STRUCTURE	186
TABLE 234: SETLIFTREQUEST JSON STRUCTURE	187
TABLE 235: SETLIFTRESPONSE JSON STRUCTURE.....	187
TABLE 236: STOPALLMOTORSRESPONSE JSON STRUCTURE.....	188
TABLE 237: TURNINPLACEREQUEST JSON STRUCTURE	189
TABLE 238: TURNINPLACERESPONSE JSON STRUCTURE.....	189
TABLE 239: ACCELDATA JSON STRUCTURE.....	190
TABLE 240: GYRODATA JSON STRUCTURE.....	190
TABLE 241: PROXDATA JSON STRUCTURE	191
TABLE 242: TOUCHDATA JSON STRUCTURE.....	191
TABLE 243: ROBOTSTATE JSON STRUCTURE.....	192
TABLE 244: ONBOARDINGPHASE ENUMERATION	193
TABLE 245: ONBOARDINGPHASESTATE ENUMERATION	193
TABLE 246: ONBOARDING JSON STRUCTURE.....	194
TABLE 247: ONBOARDING1POUCHARGINGINFOJSON STRUCTURE	194
TABLE 248: ONBOARDINGSTATE JSON STRUCTURE.....	194
TABLE 249: ONBOARDINGSTAGES ENUMERATION	194
TABLE 250: ONBOARDINGINPUTREQUEST JSON STRUCTURE.....	195
TABLE 251: ONBOARDINGSETPHASEREQUEST JSON STRUCTURE.....	195
TABLE 252: ONBOARDINGINPUTRESPONSE JSON STRUCTURE	196
TABLE 253: ONBOARDINGINPUTRESPONSE STRUCTURE	196
TABLE 254: ONBOARDINGCOMPLETERESPONSE JSON STRUCTURE.....	196
TABLE 255: ONBOARDINGPHASEPROGRESSRESPONSE STRUCTURE.....	197
TABLE 256: ONBOARDINGSETPHASERESPONSE JSON STRUCTURE	197
TABLE 257: ONBOARDINGWAKEUPRESPONSE JSON STRUCTURE	197
TABLE 258: ONBOARDINGWAKEUPSTARTEDRESPONSE JSON STRUCTURE	197
TABLE 259: ONBOARDINGSTATERESPONSE JSON STRUCTURE.....	198
TABLE 260: ONBOARDINGCOMPLETERESPONSE JSON STRUCTURE.....	198
TABLE 261: ONBOARDINGWAKEUPRESPONSE JSON STRUCTURE	198
TABLE 262: ONBOARDINGWAKEUPSTARTEDRESPONSE JSON STRUCTURE	199

TABLE 263: PHOTO_PATHMESSAGE JSON STRUCTURE	200
TABLE 264: THUMBNAIL_PATHMESSAGE JSON STRUCTURE	200
TABLE 265: PHOTO_TAKEN JSON STRUCTURE	200
TABLE 266: DELETE_PHOTOREQUEST JSON STRUCTURE	201
TABLE 267: DELETE_PHOTORESPONSE JSON STRUCTURE.....	201
TABLE 268: PHOTOREQUEST JSON STRUCTURE.....	201
TABLE 269: PHOTORESPONSE JSON STRUCTURE.....	201
TABLE 270: PHOTOSINFORESPONSE JSON STRUCTURE	202
TABLE 271: PHOTOINFO JSON STRUCTURE	202
TABLE 272: THUMBNAILREQUEST JSON STRUCTURE	203
TABLE 273: THUMBNAILRESPONSE JSON STRUCTURE	203
TABLE 274: ACCOUNTSETTING JSON STRUCTURE	204
TABLE 275: UPDATESETTINGSREQUEST JSON STRUCTURE.....	204
TABLE 276: UPDATESETTINGSRESPONSE JSON STRUCTURE	204
TABLE 277: JSON PARAMETERS FOR UPDATESACCOUNTSETTINGSREQUEST.....	205
TABLE 278: UPDATESACCOUNTSETTINGSRESPONSE JSON STRUCTURE.....	205
TABLE 279: UPDATESSTATUS ENUMERATION.....	206
TABLE 280: CHECKUPDATESSTATUSRESPONSE JSON STRUCTURE.....	207
TABLE 281: UPDATESANDRESTARTRESPONSE JSON STRUCTURE.....	207
TABLE 282: THE CLOUD SERVICES CONFIGURATION FILE	208
TABLE 283: JSON PARAMETERS FOR JDOC REQUEST	209
TABLE 284: LOG UPLOAD HTTP HEADER FIELDS	211
TABLE 285: CRASH UPLOAD FORM FIELDS	211
TABLE 286: DAS MANAGER SQS KEY-VALUE PAIRS	211
TABLE 287: THE CAMERA CALIBRATION JSON STRUCTURE	217
TABLE 288: THE VISION PROCESSES	217
TABLE 289: THE CUBE LEDs JSON STRUCTURE	236
TABLE 290: THE CUBE LEDs PATTERN STRUCTURE.....	237
TABLE 291: THE JSON STRUCTURE.....	250
TABLE 292: THE SPEEDTRAITS JSON STRUCTURE	250
TABLE 293: THE JSON STRUCTURE.....	251
TABLE 294: THE JSON STRUCTURE.....	251
TABLE 295: THE SMARLING JSON STRUCTURE	251
TABLE 296: THE JSON STRUCTURE.....	252
TABLE 297: ANIMCLIPS STRUCTURE	257
TABLE 298: ANIMCLIP STRUCTURE	257
TABLE 299: AUDIOEVENTGROUP STRUCTURE.....	257
TABLE 300: AUDIOPARAMETER STRUCTURE.....	258
TABLE 301: AUDIOSTATE STRUCTURE.....	258
TABLE 302: AUDIOSWITCH STRUCTURE	258
TABLE 303: BACKPACKLIGHTS STRUCTURE	258
TABLE 304: BODYMOTION STRUCTURE	259
TABLE 305: EVENT STRUCTURE.....	259
TABLE 306: FACEANIMATION STRUCTURE	259
TABLE 307: HEADANGLE STRUCTURE	259
TABLE 308: LIFTHEIGHT STRUCTURE	260
TABLE 309: KEYFRAMES STRUCTURE	260
TABLE 310: PROCEDURALFACE STRUCTURE	261
TABLE 311: RECORDHEADING STRUCTURE.....	262
TABLE 312: ROBOTAUDIO STRUCTURE	262
TABLE 313: TURNTORECORDEDHEADING STRUCTURE.....	262
TABLE 314: CHECKUPDATESSTATUSRESPONSE JSON STRUCTURE.....	267
TABLE 315: EMOTIIONEVENT JSON STRUCTURE	267

TABLE 316: EMOTIAFFECTOR JSON STRUCTURE	267
TABLE 317: THE EMOTION EVENT NAMES.....	267
TABLE 318: THE OWNERS ACCOUNT INFORMATION.....	274
TABLE 319: BUTTONWAKEWORD ENUMERATION.....	275
TABLE 320: EYECOLOR ENUMERATION.....	275
TABLE 321: THE EYE COLOUR JSON STRUCTURE.....	276
TABLE 322: VOLUME ENUMERATION	276
TABLE 323: THE ROBOTSETTINGSCONFIG JSON STRUCTURE	276
TABLE 324: THE SETTING CONTROL STRUCTURE.....	277
TABLE 325: THE COZMO ACCOUNT SETTINGS.....	278
TABLE 326: THE FEATURE FLAG STRUCTURE.....	279
TABLE 327: MANIFEST.INI META SECTION.....	281
TABLE 328: MANIFEST.INI IMAGE STREAM SECTIONS.....	282
TABLE 329: UPDATE-ENGINE STATUS FILE	283
TABLE 330: VECTOR DIAGNOSTIC & LOGGING SOFTWARE.....	285
TABLE 331: FILES IN THE LOG ARCHIVE.....	289
TABLE 332: THE CONSOLE FILTER CHANNEL STRUCTURE	289
TABLE 333: THE CHANNEL LOGGING ENABLE STRUCTURE.....	290
TABLE 334: THE LOGGING LEVEL ENABLE STRUCTURE.....	290
TABLE 335: THE CHANNELS	290
TABLE 336: THE WiFi RELATED STATS /PROC FILES	292
TABLE 337: NAMED DEVICE AND CONTROL FILES	293
TABLE 338: THE EXPERIMENTS TBD STRUCTURE	293
TABLE 339: THE META JSON STRUCTURE	293
TABLE 340: THE EXPERIMENT JSON STRUCTURE	294
TABLE 341: THE VARIATION JSON STRUCTURE.....	294
TABLE 342: COMMON ACRONYMS AND ABBREVIATIONS	299
TABLE 343: GLOSSARY OF COMMON TERMS AND PHRASES	301
TABLE 344: TOOLS USED BY ANKI	304
TABLE 345: ALEXA FILES	308
TABLE 346: THE SYSTEM FAULT CODES	310
TABLE 347: OTA UPDATE-ENGINE STATUS CODES.....	310
TABLE 348: THE FILE SYSTEM MOUNT TABLE	312
TABLE 349: THE PARTITION TABLE	312
TABLE 350: FILES	314
TABLE 351: NAMED DEVICE AND CONTROL FILES	315
TABLE 352: THE BLUETOOTH LE SERVICES	317
TABLE 353: THE CUBE's DEVICE INFO SETTINGS	317
TABLE 354: CUBE's ACCELEROMETER SERVICE CHARACTERISTICS.....	318
TABLE 355: VECTOR's BLUETOOTH LE SERVICES.....	318
TABLE 356: THE VECTOR's DEVICE INFO SETTINGS.....	318
TABLE 357: VECTOR's SERIAL SERVICE CHARACTERISTICS	319
TABLE 358: THE SERVERS THAT VECTOR CONTACTS.....	320
TABLE 359: THE SERVERS THAT THE MOBILE APPLICATION CONTACTS.....	320
TABLE 360: THE AMAZON ALEXA VOICE SERVICE SERVERS THAT VECTOR CONTACTS.	320
TABLE 361: THE FEATURES.....	322
TABLE 362: THE "HEY VECTOR" PHRASES	324
TABLE 363: THE VECTOR QUESTIONS PHRASES	326
TABLE 364: VERSION INFO, POSTED TO DAS	328
TABLE 365: START UP INFORMATION, POSTED TO DAS.....	328
TABLE 366: SETTINGS AND PREFERENCES, POSTED TO DAS.....	329
TABLE 367: POWER MANAGEMENT EVENTS, POSTED TO DAS	329
TABLE 368: BATTERY LEVEL EVENTS AND STATISTICS.....	329

TABLE 369: CHARGER STATISTICS AND EVENTS, POSTED TO DAS	330
TABLE 370: MOTOR EVENTS.....	330
TABLE 371: BASE-BOARD / SPINE RELATED DAS EVENTS.....	330
TABLE 372: ACCESSORY CUBE RELATED DAS EVENTS.....	331
TABLE 373: MOBILE APPLICATION / SDK RELATED DAS EVENTS.....	331
TABLE 374: UPDATE EVENTS	331
TABLE 375: BEHAVIOUR, FEATURE, MOOD AND ENGINE RELATED DAS EVENTS.....	332
TABLE 376: THE ROBOT LIFETIME STATS SCHEMA.....	332

Examples

EXAMPLE 1: BLUETOOTH LE ENCRYPTION STRUCTURES	81
EXAMPLE 2: BLUETOOTH LE KEY PAIR	81
EXAMPLE 3: BLUETOOTH LE ENCRYPTION & DECRYPTION KEYS	81
EXAMPLE 4: DECRYPTING A BLUETOOTH LE MESSAGE	82
EXAMPLE 5: ENCRYPTING A BLUETOOTH LE MESSAGE.....	82
EXAMPLE 6: COMPUTING THE CAMERA POSE SOURCE: ANKI.....	219
EXAMPLE 7: CHECKING THE MANIFEST.INI SIGNATURE.....	281
EXAMPLE 8: DECRYPTING THE OTA UPDATE ARCHIVES.....	282
EXAMPLE 9: DECRYPTING THE OTA UPDATE ARCHIVES WITH OPEN SSL 1.1.0 AND LATER	282

Equations

EQUATION 1: RELATIONSHIP BETWEEN FIELD OF VIEW AND FOCAL LENGTH	216
EQUATION 2: CAMERA CALIBRATION MATRIX	217
EQUATION 3: TREAD SPEEDS BASED ON ARC RADIUS	255

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 discussed 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 hardest 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 ELECTRONICS DESIGN. An overview of the Vector's electronics design.
- CHAPTER 3: HEAD-BOARD ELECTRONICS DESIGN. A detailed look at the electronics design of Vector's main processing board.
- CHAPTER 4: BACKPACK & BASE-BOARD ELECTRONICS DESIGN. A detailed look at the electronics design of Vector's backpack and motor driver boards.
- CHAPTER 5: ACCESSORY ELECTRONICS DESIGN. A look at the electronics design of Vector's accessories.

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

- **CHAPTER 6: ARCHITECTURE.** A detailed look at Vector's overall software architecture.
- **CHAPTER 7: STARTUP.** A detailed look at Vector's startup, and shutdown processes
- **CHAPTER 8: POWER MANAGEMENT.** A detailed look at Vector's architecture for battery monitoring, changing and other power management.
- **CHAPTER 9: BUTTON & TOUCH INPUT AND OUTPUT LEDs**
- **CHAPTER 10: AUDIO INPUT**
- **CHAPTER 11: MOTION SENSING**

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

- **CHAPTER 12: COMMUNICATION.** A look at Vector's communication stack.
- **CHAPTER 13: BLUETOOTH LE.** The Bluetooth LE protocol that Vector responds to.
- **CHAPTER 14: SDK PROTOCOL.** The HTTPS protocol that Vector responds to.
- **CHAPTER 15: CLOUD.** A look at how Vector syncs with remote services.

PART IV: ADVANCED FUNCTIONS. This part describes items that are Vector's primary function.

- **CHAPTER 16: IMAGE PROCESSING.** Vector vision system is sophisticated, with the ability to recognize marker, faces, and objects; to take photographs, and acts as a key part of the navigation system.
- **CHAPTER 17: MAPPING & NAVIGATION.** A look at Vector's mapping and navigation systems.
- **CHAPTER 18: ACCESSORIES.** A look at Vector's home (charging station), companion cube and custom objects.

PART V: ANIMATION. Vector uses animations – “sequence[s] of highly coordinated movements, faces, lights, and sounds” – “to demonstrate an emotion or reaction.” This part describes how the animation system works.

- **CHAPTER 19: ANIMATIONS.** An overview how Vector's scripted animations represents the “movements, faces, lights and sounds;” and how they are coordinated.
- **CHAPTER 20: DISPLAY & PROCEDURAL FACE.** Vector displays a face to convey his mood and helps forms an emotional connection with his human.
- **CHAPTER 21: AUDIO PRODUCTION.** A look at how Vector's sound effects and how he speaks
- **CHAPTER 22: MOTION CONTROL.** A look at how Vector's moves.
- **CHAPTER 23: ANIMATION FILE FORMAT.** The format of Vector's binary animation file

PART VI: HIGH-LEVEL AI.

- **CHAPTER 24: BEHAVIOR.** A look at how Vector behaviors, emotions, etc.

PART VI: MAINTENANCE. This part describes items that are not Vector's primary function; they are practical items to support Vector's operation.

- CHAPTER 25: SETTINGS, PREFERENCES, FEATURES AND STATISTICS. A look at how Vector syncs with remote servers
- CHAPTER 26: SOFTWARE UPDATES. How Vector's software updates are applied.
- CHAPTER 27: 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.

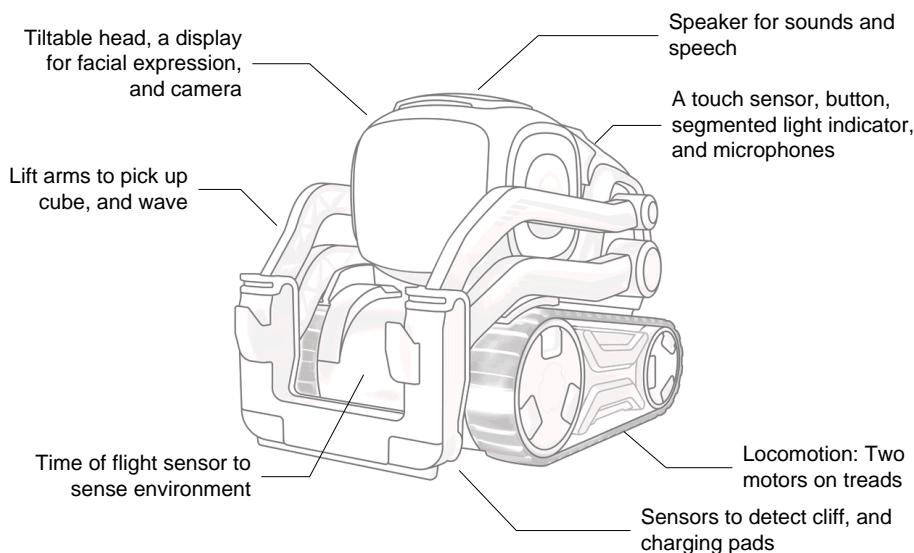


Figure 1: Vector's main features

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 name¹, 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 to:

- 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.

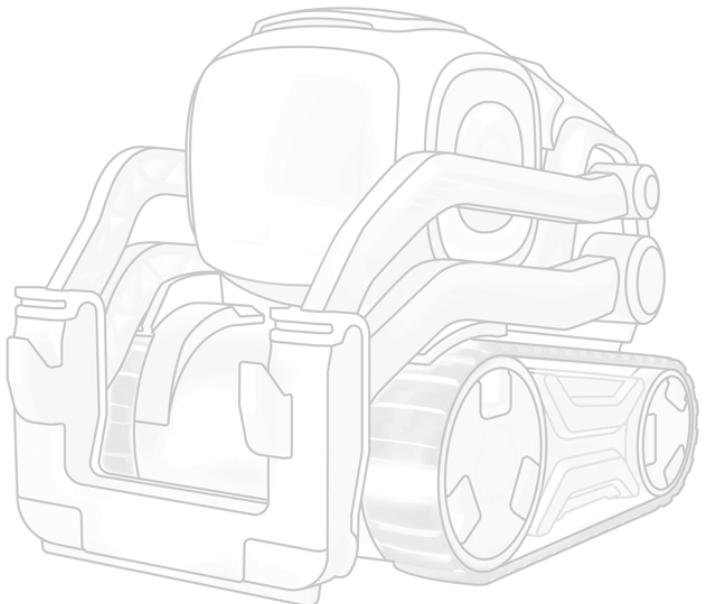
⁶ <https://aws.amazon.com/blogs/iot/introducing-alexa-voice-service-integration-for-aws-iot-core/>
Alexa’s SDK and services have continued to evolve. New Alexa SDKs allow simpler processors and smaller code by acting as little more than a remote microphone.

PART I

Electronics Design

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

- VECTOR'S ELECTRONICS DESIGN. An overview of the Vector's electronics design.
- HEAD-BOARD ELECTRONICS DESIGN. A detailed look at the electronics design of Vector's main processing board.
- BACKPACK & BASE-BOARD ELECTRONICS DESIGN. A detailed look at the electronics design of Vector's backpack and motor driver boards.
- 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 design of Vector's electronics:

- Design Overview outlining the main subsystems
- Power distribution

Subsequent chapters will examine in detail the design of the subsystems

7. DESIGN OVERVIEW

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

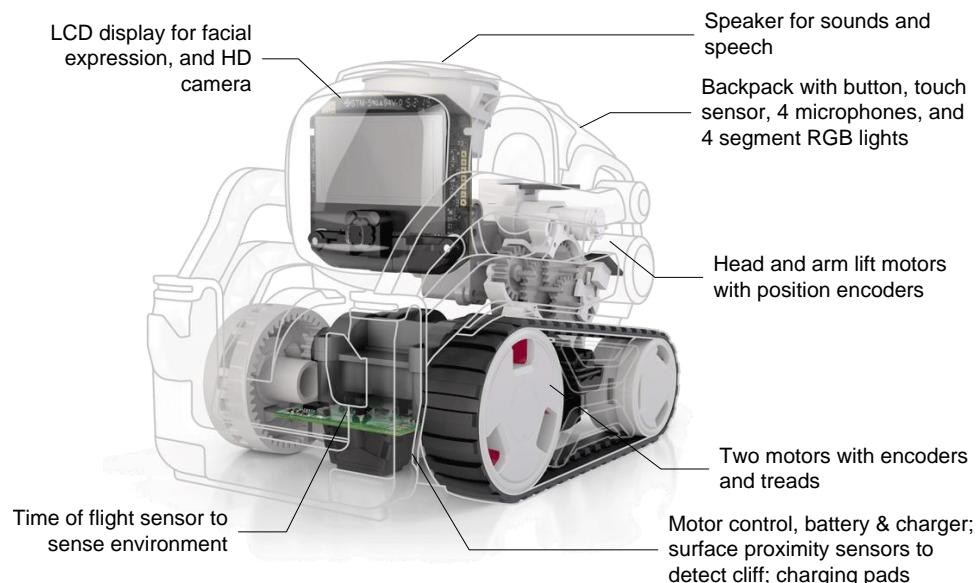


Figure 2: Vector's main elements

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

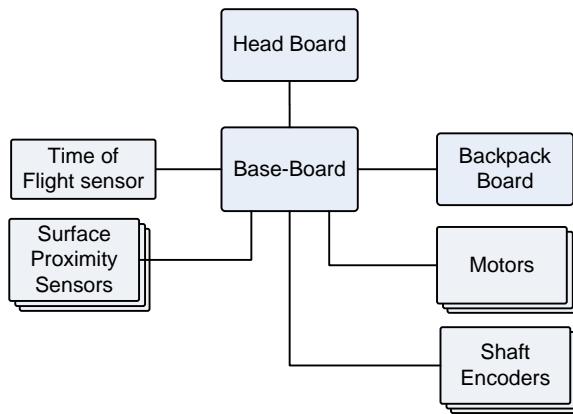


Figure 3: Circuit board topology

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

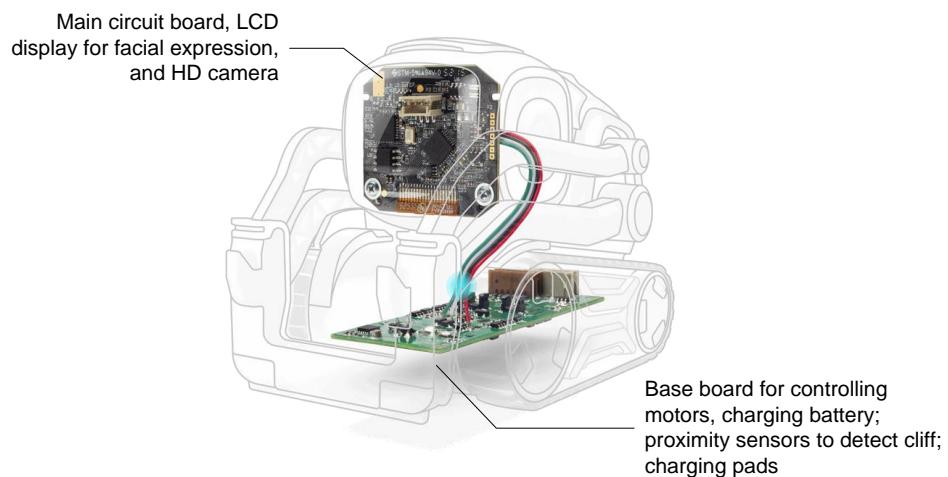


Figure 4: Vector's main microcontroller circuit boards

The table below summarizes the boards:

Circuit Board	Description
backpack board	The backpack board has 4 RGB LEDs, 4 MEMS microphones, a touch wire, and a button. This board connects to the base-board.
base-board	Drives the motor, power management battery charger
encoder-boards	The two encoder boards have 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.

Table 2: Vector's circuit boards

7.1. POWER SOURCE AND DISTRIBUTION TREE

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

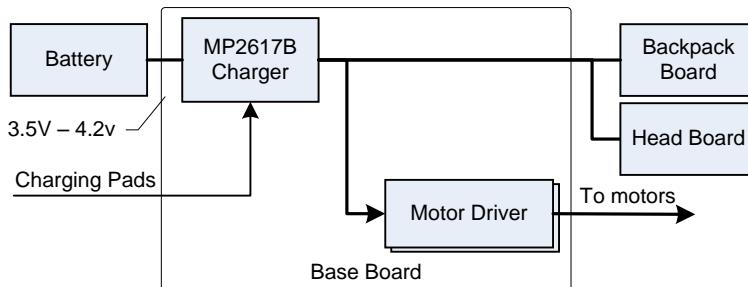


Figure 5: Power distribution

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.

CHAPTER 3

Head-board

Electronics Design

Description

This chapter describes the electronic design of Vector's head-board:

- Detailed design of the head-board

8. 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.

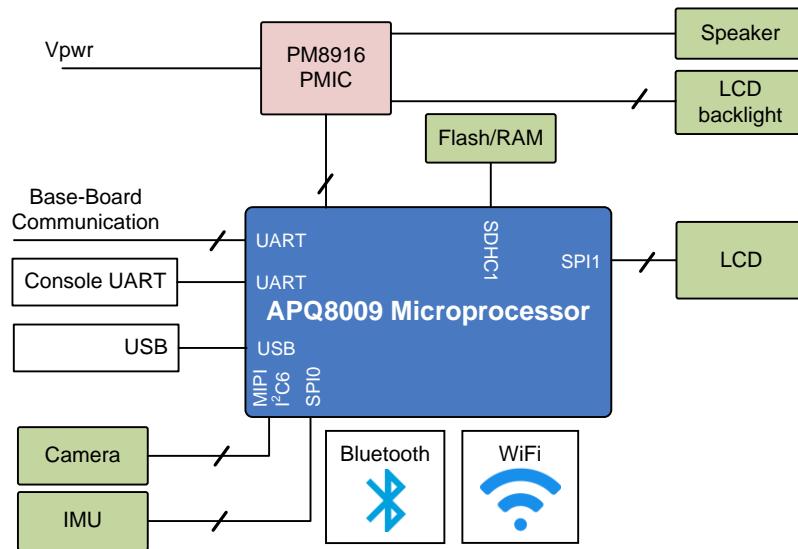


Figure 6: Head-board block diagram

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.
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

Table 3: The head-boards functional elements

8.1. 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.

8.2. SPEAKERS

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

8.3. 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.

GPIO	Description	Table 4: The camera controls
26	Camera interface clock	
48	Camera reset	
83	Camera power enable (from PM8916 PMIC)	
94	Camera standby	

8.4. THE LCD

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.

LCD display

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 LCD's illuminated rows were regularly alternated to give a retro-technology interlaced row effect, like old CRTs.

US Patent 20372659

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

8.5. 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 calibrated⁷

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

8.6. 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.

⁷ Todo: look into what this means. R43 on the base-board 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?

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

9. REFERENCES & RESOURCES

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

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

CHAPTER 4

Backpack & Base-board Electronics

Design Description

This chapter describes the electronic design of the Anki Vector's supplemental boards:

- Detailed design of the backpack-board
- Detailed design of the base-board
- Power characteristics

10. THE BACKPACK BOARD

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

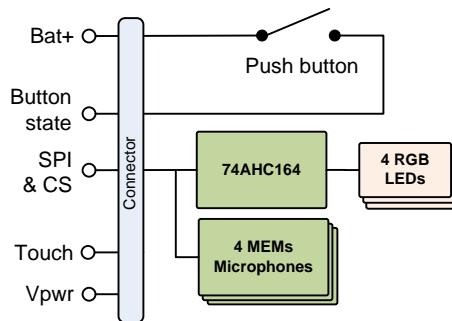


Figure 7: Backpack board block diagram

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

Elements	Description	Table 5: Backpack board functional elements
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)	

10.1. 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

10.2. 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).

10.2.1 The LED controls

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.

Backpack LED control scheme

- 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, D4.⁸
- 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).

⁸ 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

11. 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.

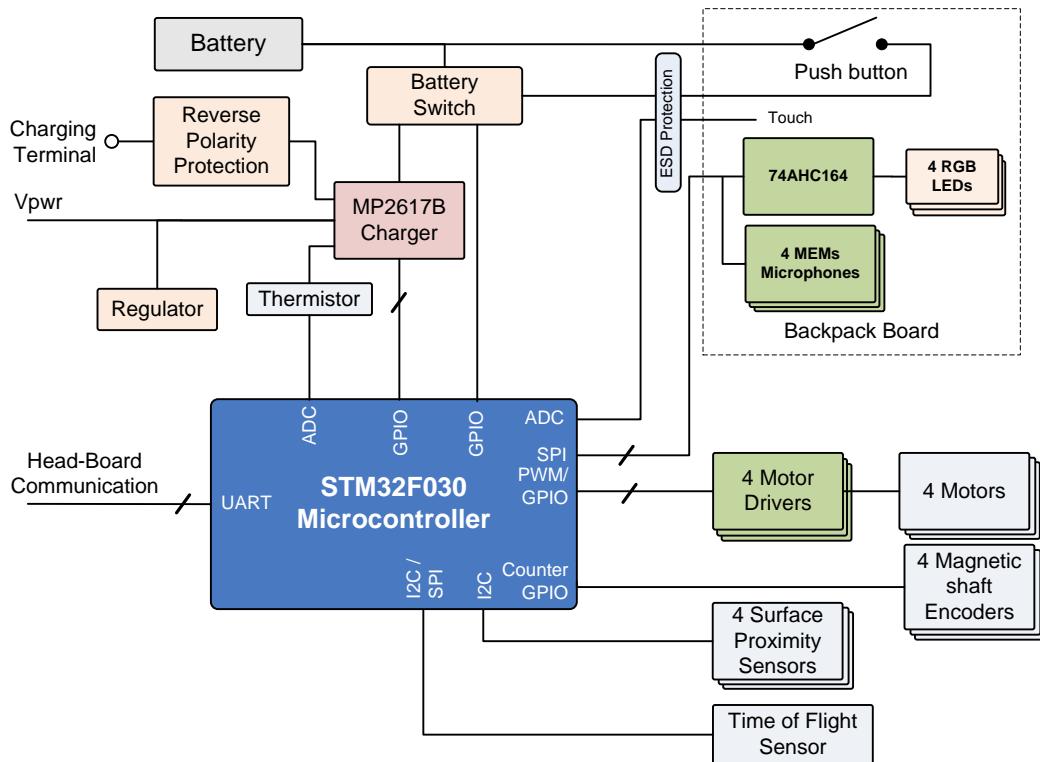


Figure 8: Base-board block diagram

The functional elements of the base-board are:

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 two 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

Table 6: The base-board functional elements

	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 ⁹	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 ² C.

11.1. 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.

11.1.1 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.

11.1.2 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 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.

⁹ Not identified. The customer service screen does show a battery pack temperature, indicating that this is reported.

11.1.3 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 use. This constraint shapes the battery disconnect-reconnect logic. The schematic below shows one way to do this:

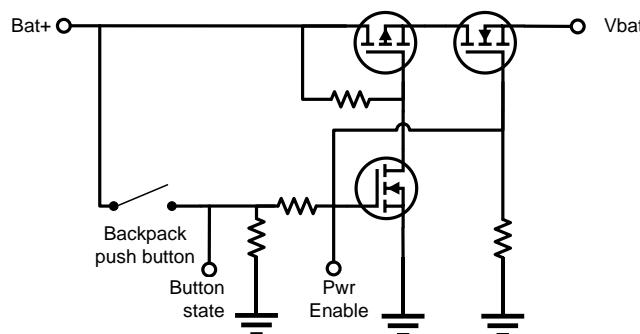


Figure 9: A representative battery connect switch

Two MOSFETs (a PFET and NFET)¹⁰ 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 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.

11.1.4 Charging

The charging station pads are connected to a MP2617B charger IC thru a reverse polarity protection circuit. The reverse polarity protection¹¹ is a DMG2305UX PFET in a diode

charging station pads

¹⁰ Q11 and/or Q12

¹¹ Q14

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

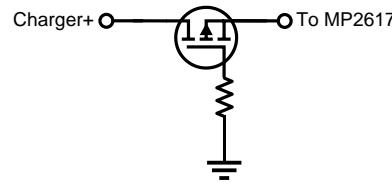


Figure 10: A representative PFET based reversed polarity protection

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.

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.

supplying power from the charging station

charging states

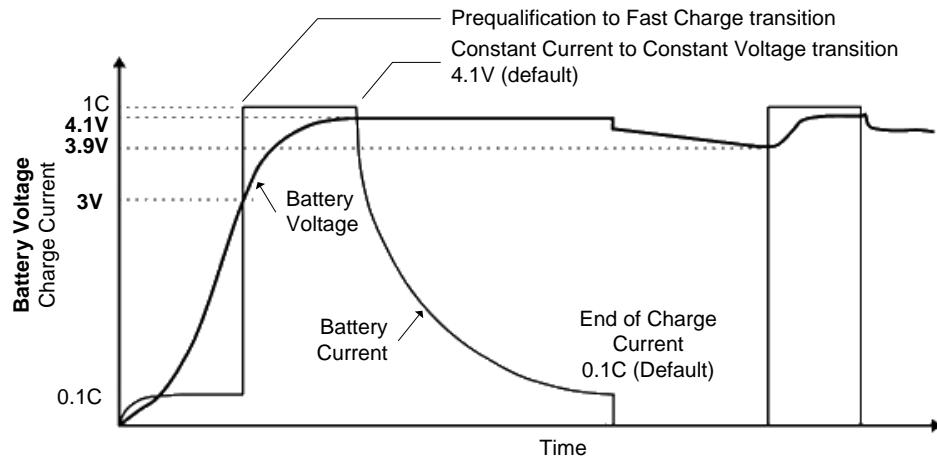


Figure 11: Charging profile (adapted from Texas Instruments)

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 converters. 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.¹² 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.¹³

11.1.5 Brown-out

The motor stall current is enough to cause Vector to brown-out and shut down unexpectedly.

motor stall & brown out effects

This indicates two possible mechanisms:

- 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.

11.2. 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.

11.3. STM32F030 MICROCONTROLLER

The base-board is controlled by a STM32F030C8T6 microcontroller (MCU), in a LQFP48. 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

¹² Other reports suggest up to 2As into the battery, possible with the use of high-power USB adapters intended to support tablet recharge.

¹³ 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 crystal¹⁴ and uses an internal RC oscillator instead.

11.3.1 Manufacturing test connector

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

11.3.2 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 be analyzed 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.

11.4. SENSING

11.4.1 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 I²C.

11.4.2 Proximity sensing

Has 4 IR proximity sensors that are 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 I²C, the brightness sensed by the detector. This are often pulsed, to reject sunlight; and use a configurable threshold to reduce sensitivity to ambient light.

¹⁴ as far as I can see

11.4.3 Touch sensing

The touch sensing works by alternating pulsing and sampling (with the ADC) the touch wire.

Anki SDK

The samples will vary “by various environmental factors such as whether the robot is on its charger, being held, humidity, etc.”

11.5. MOTOR DRIVER AND CONTROL

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

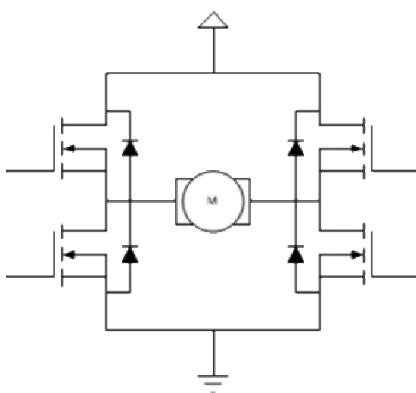


Figure 12: Motor driver H-bridge

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.

11.6. COMMUNICATION

The base-board communicates with the head-board via RS-232 3.3V (3 Mbits/sec¹⁵). 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.

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:

SPI communication with 2 microphones simultaneously

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.

¹⁵ 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 at a rate of 15625 samples/sec.

11.7. CONNECTORS & TEST POINTS

The base-board has the following connectors:

Alexander Entinger

- Connector to the head-board
- Connector to the head motor & encoder
- Connector to the lift motor & encoder
- Connector to the time of flight sensor

11.7.1 The head-motor connector

P1 Head motor connector (P1)

Pin#	Label	Test point	Cable Color	Description	
1	CA1	HENCK	Brown	Head encoder clock	Table 7: Head Motor Connector (P1) pin map
2	E2	HENCB	Yellow	Head encoder output B	
3	E1	HENCA	Green	Head encoder output A	
4	V _{DD}		White	Head encoder voltage source	
5	Motor -		Black	Motor connection	
6	Motor +		Red	Motor connection	

11.7.2 The lift-motor connector

Lift motor connector (P2)

Pin#	Label	Test point	Cable Color	Description	
1	CA1	LENCK	Brown	Lift encoder clock	Table 8: Lift Motor Connector (P2) pin map
2	E2	B	Yellow	Lift encoder output B	
3	E1	A	Green	Lift encoder output A	
4	V _{DD}		White	Head encoder voltage source	
5	Motor -		Black	Motor connection	
6	Motor +		Red	Motor connection	

11.7.3 The lift-motor connector

Front (Time of Flight) sensor connector

Pin#	Label	Test point	Cable Color	Description
1	V _{DD}	V _{DD} (TP)	Red	Sensor voltage source
2	SCL1		Yellow	I ² C serial clock
3	SDA1		Green	I ² C serial data in/out
4	GND	GND (DC)	Black	Sensor ground reference

Table 9: Front Sensor (time of flight) Connector pin map

11.7.4 The debug connector

Debug connector

Pin#	Label	Description
1	VX	External power supply (connected with charger connector positive.)
2	BAI	Internal power supply for head
3	TX	UART transmit; connects to STM32F030C8T6 Pin #12 (PA2 = USART2_TX)
4	SWCLK	Single wire debug clock signal
5	SWDIO	Single wire debug bi-directional data signal
6	NRST	Processor reset (reset is transition from low to high).
7	GND	Ground

Table 10: Debug Connector pin map

11.7.5 Other base-board test points

Other test points

Test Point	Layer	Description
V _{dd}	Bottom	Head encoder clock
BODY_TX	Bottom	Head encoder output B
SCL2	Top	I ² C serial clock
SDA2	Top	I ² C serial data in/out

Table 11: Head Motor Connector (P1) pin map

12. 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
- Everlight *EAAPMST3923A2*
- 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>
- 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. VL53L1 Long distance ranging Time-of-Flight sensor
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/resource/en/datasheet/vl53l1x.pdf>

CHAPTER 5

Accessory Electronics

Design Description

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

- The charging station
- The companion cube

13. CHARGING STATION

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

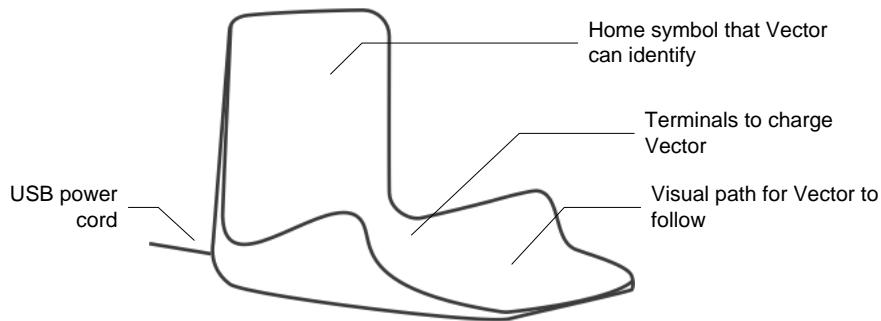


Figure 13: Charging station main features

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 18).

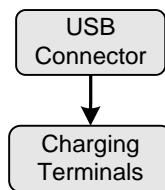


Figure 14: Charging station block diagram

14. 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 18).

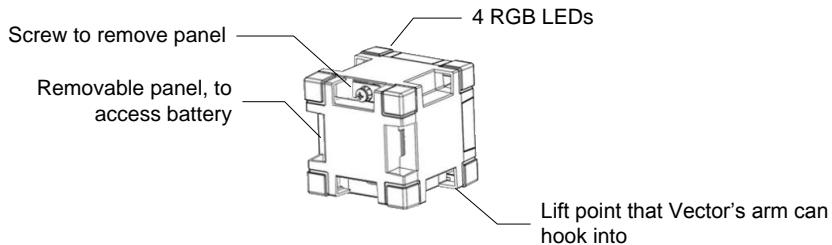


Figure 15: Cube's main features

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:

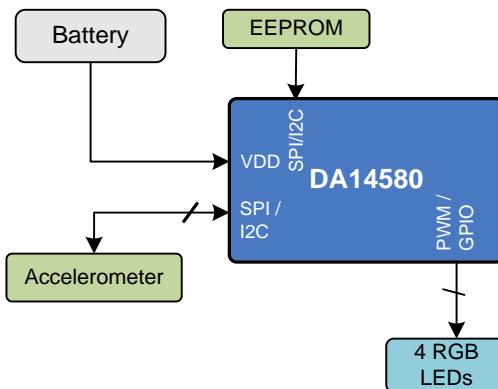


Figure 16: Block diagram of the Cube's electronics

The Cube's electronic design includes the following elements:

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

Table 12: The Cube's electronic design elements

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.

14.1. 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.

14.2. 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.
- MOTION SENSING



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

CHAPTER 6

Architecture

This chapter describes Vector's software architecture:

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

15. OVERVIEW OF VECTOR'S COMMUNICATION INFRASTRUCTURE

Vector's architecture has a structure something like:

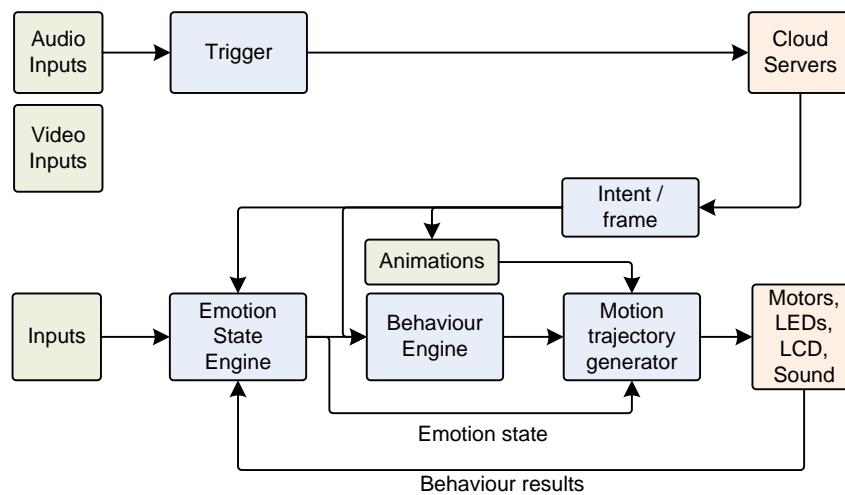


Figure 17: The overall functional block diagram

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

15.1. APPLICATION SERVICES ARCHITECTURE

The application is divided into the following services

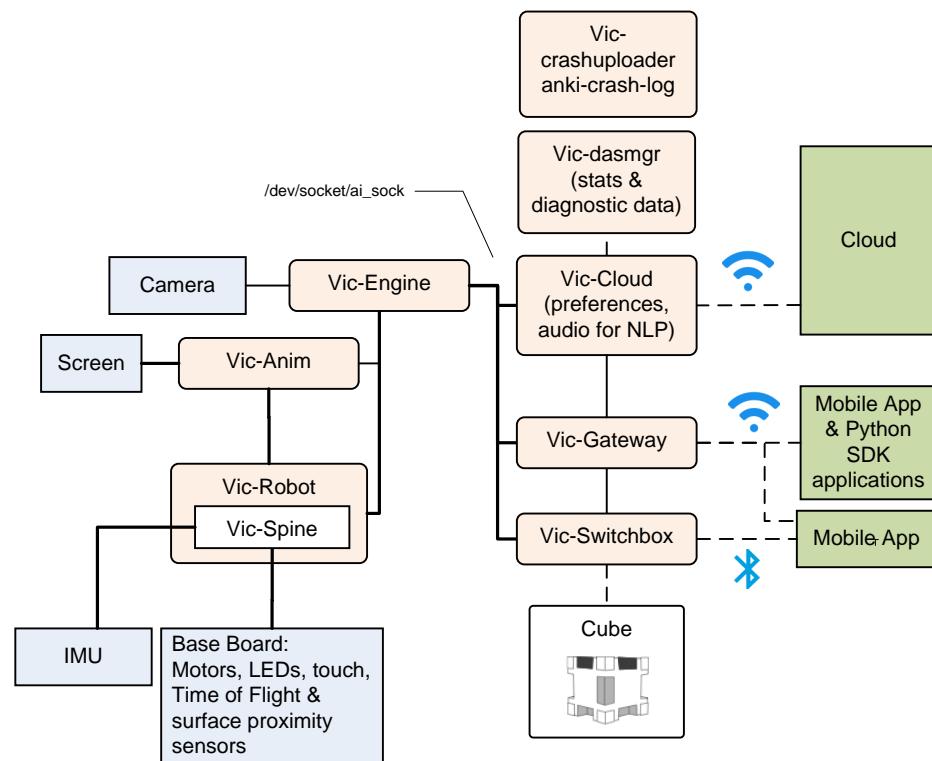


Figure 18: The overall communication infrastructure

There are multiple applications that run:

Services	Speculated purpose
vic-anim	Probably plays multi-track animations (which include motions as well as LCD display and sound) config file: /anki/etc/config/platform_config.json /anki/data/assets/cozmo_resources/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 13). Routes messages between the other

Table 13: Vector processes

15.2. 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 they 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.

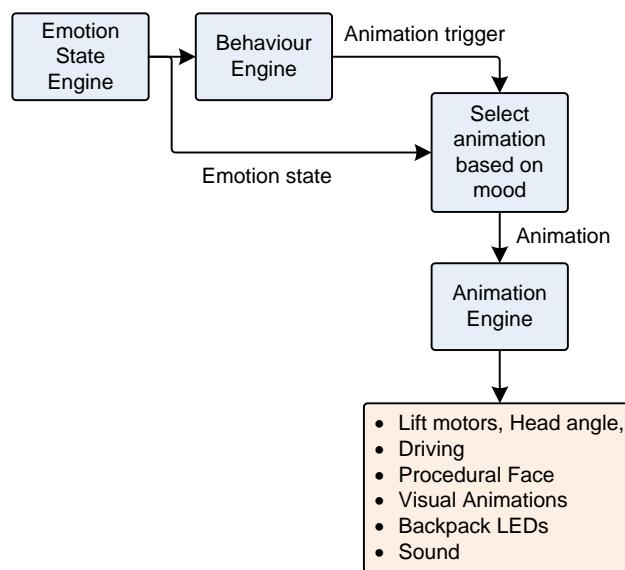


Figure 19: The behaviour-animation flow

16. 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 7 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 7 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.

16.1. ELECTRONIC MEDICAL RECORDS (EMR)

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

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

Table 14: Electronic Medical Record (EMR)

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

16.2. 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.
<i>cloud</i>	Holds the SDK TLS certificate and signing keys. With newer firmware, the folder may also hold some other calibration information.
<i>nvStorage</i>	holds some binary ".nvdata" files

Table 15: OEM partition file hierarchy

17. SECURITY AND PRIVACY

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:

Anki Security & Privacy Policy

- 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 faces¹⁷ 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.)

17.1. ENCRYPTED FILESYSTEM

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

17.2. 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 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.

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.

17.2.1 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.

17.3. 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)

18. 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.

18.1. CONFIGURATION FILES

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
<code>DataPlatformCachePath</code>	<code>"/data/data/com.anki.victor/cache"</code>	
<code>DataPlatformPersistentPath</code>	<code>"/data/data/com.anki.victor/persistent"</code>	

Table 16: The platform config JSON structure

<i>DataPlatformResourcesPath</i>	“/anki/data/assets/cozmo_resources”	The path to most configuration files and assets
----------------------------------	-------------------------------------	---

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.

19. SOFTWARE-HARDWARE LAYERS

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

19.1. THE BASE BOARD INPUT/OUTPUT

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

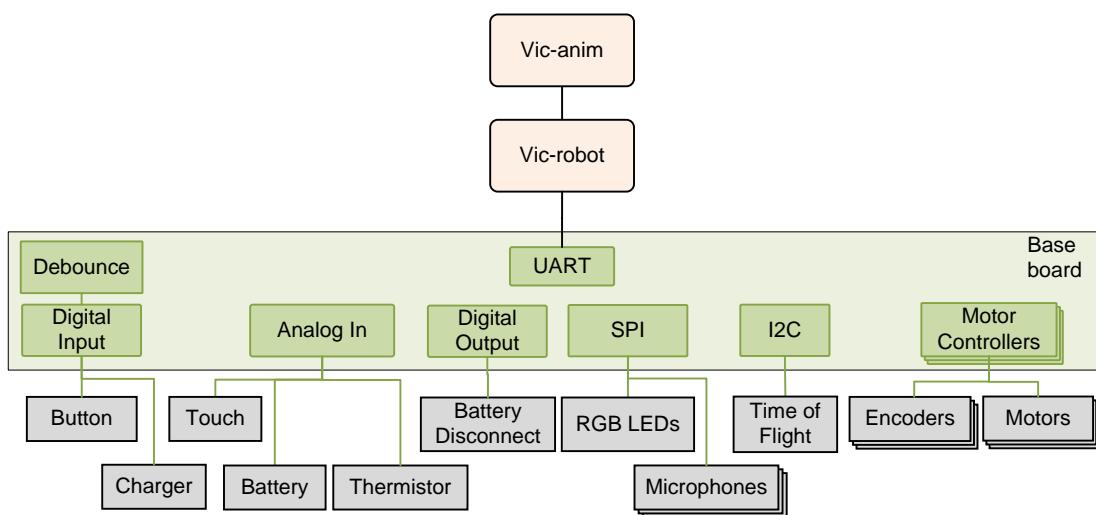


Figure 20: The baseboard-related architecture

19.2. THE LCD DISPLAY

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

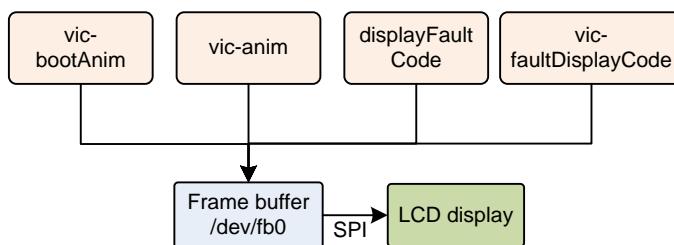


Figure 21: The LCD architecture

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) to the display.

`vic-animate` 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-animate` have a visual aesthetic is unlike the rest of Vector. These modes employ a barebones system for the display.

The text appears to be 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.

19.3. THE CAMERA

The camera subsystem has the following architecture:

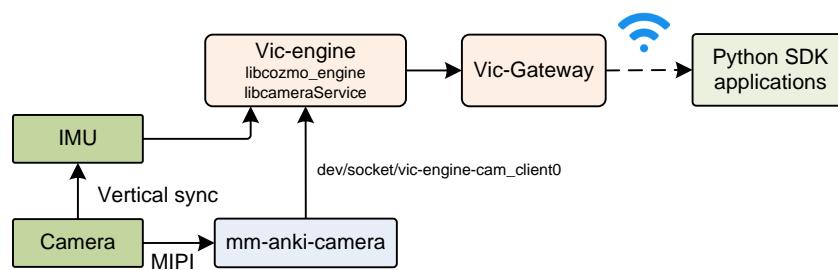


Figure 22: The camera architecture

Daniel Casner, 2019
Embedded Vision
Summit

The camera's vertical synchronization signal is connected 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 (see chapter 17) 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.

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.)

20. 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-socts/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 7

Startup

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

- The startup process
- The shutdown steps

21. 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 respond 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.

21.1. QUALCOMM'S PRIMARY AND SECONDARY BOOTLOADER

Meanwhile, on the head-board:

1. “Qualcomm’s Primary Boot Loader is verified and loaded into [RAM] memory¹⁹ 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. Roe 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.

21.2. 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.
Roe 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 26 where updating `aboot`, `boot`, and `system` partitions are discussed.

21.3. 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 (chrony), 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

21.4. 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 27 for a description of the display of fault codes and diagnostics. See Appendix C for fault codes.

22. 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.

22.1. TURNING VECTOR OFF (INTENTIONALLY)

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

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(?)

Table 17: Vector shutdown codes

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

22.2. 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.

22.3. 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.

23. 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 about 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/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 8

Power management

This chapter describes Vector's power management:

- The battery management
- Load shedding
- Charger info

24. POWER MANAGEMENT

24.1. 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.

24.1.1 Battery levels

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

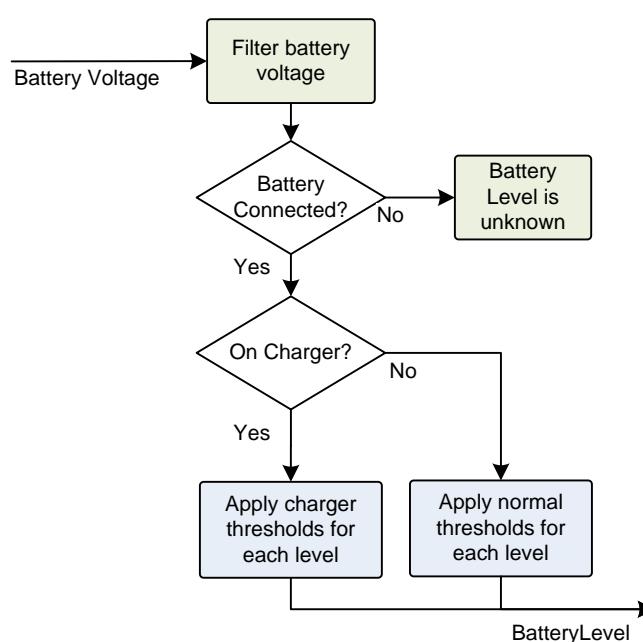


Figure 23: The battery level classification tree

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:

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

Table 18:
`BatteryLevel` codes²⁴
as they apply to
Vector

The battery levels are organized conventionally:

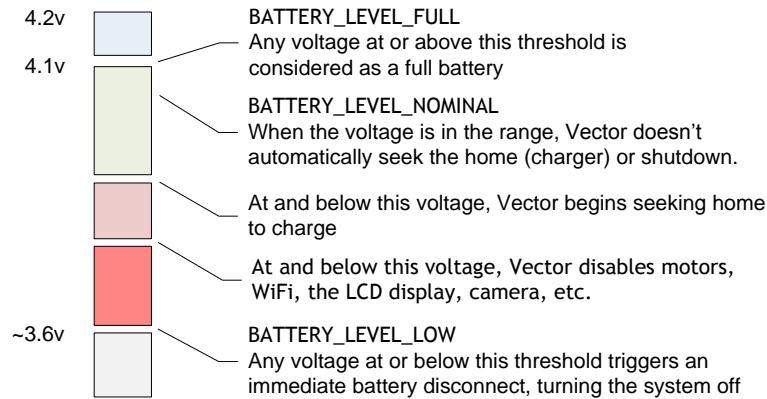


Figure 24: The battery thresholds

The current battery level and voltage can be requested with the Battery State command (see Chapter 14, *Battery State*). The response will provide the current battery voltage, and interpreted level.

24.2. 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.

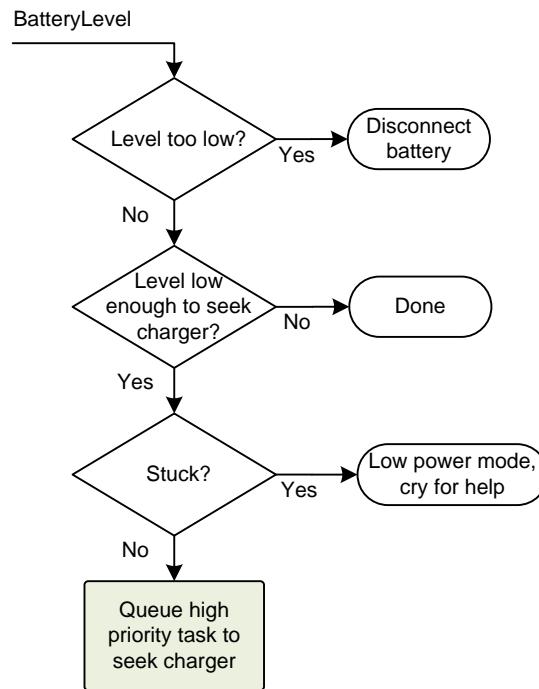


Figure 25: The response to battery level

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; presumably the time of flight sensor as well.
- He reduces processing on the processor

24.2.1 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 14 for details.) Vector is in a calm power model if the ROBOT_STATUS_CALM_POWER_MODE bit is set (in the status value).

25. CHARGING

Tracks whether 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 14 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 14, *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 9

Basic Inputs and Outputs

This chapter describes Vector's most basic input and output: his button, touch and LEDs:

- Touch and button input
- Backpack Lights control

26. 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. Four 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:

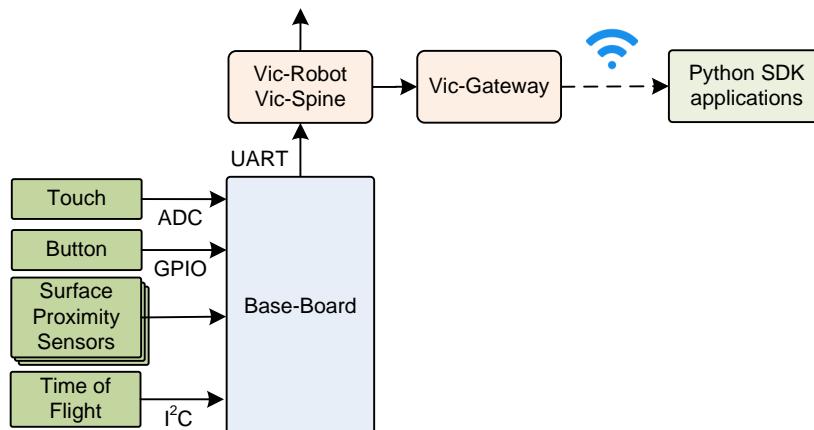


Figure 26: The touch and button input architecture

The state of the inputs (button, touch, surface proximity and time of flight sensors) are reported in the `RobotStatus` message. (See chapter 14 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 being 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.

27. BACKPACK LIGHTS CONTROL

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

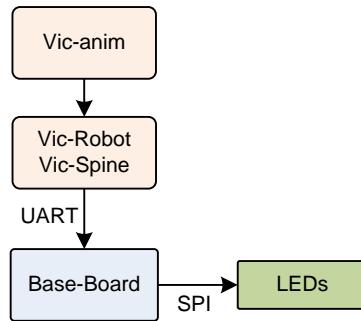


Figure 27: The backpack lights output architecture

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 19: The Backpack LEDs JSON structure

Field	Type	Units	Description
<code>offColors</code>	array of 3 colors	<code>RGBA</code>	Each color corresponds to each of the 3 lower back pack lights. Each color is represented as an array of 4 floats (red, green, blue, and alpha), in the range 0..1. Alpha is always 1.
<code>offPeriod_ms</code>	<code>float[3]</code>	<code>ms</code>	The “off” duration for each of the 3 back pack lights. This is the duration to show each cube light in its corresponding “off” color (in <code>offColors</code>).
<code>offset</code>	<code>float[3]</code>		always 0
<code>onColors</code>	array of 3 colors	<code>RGBA</code>	Each color corresponds to each of the 3 lower back pack lights. Each color is represented as an array of 4 floats (red, green, blue, and alpha), in the range 0..1. Alpha is always 1.
<code>onPeriod_ms</code>	<code>float[3]</code>	<code>ms</code>	The “on” duration for each of the 4 cube lights. This is the duration to show each cube light in its corresponding “on” color (in <code>onColors</code>).
<code>transitionOffPeriod_ms</code>	<code>float[3]</code>	<code>ms</code>	The time to transition from the on color to the off color.
<code>transitionOnPeriod_ms</code>	<code>float[3]</code>	<code>ms</code>	The time to transition from the off color to the on color.

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. See Chapter 18, *Cube Animations*.

CHAPTER 10

Audio Input

This chapter describes the sound input system:

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

28. AUDIO INPUT

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

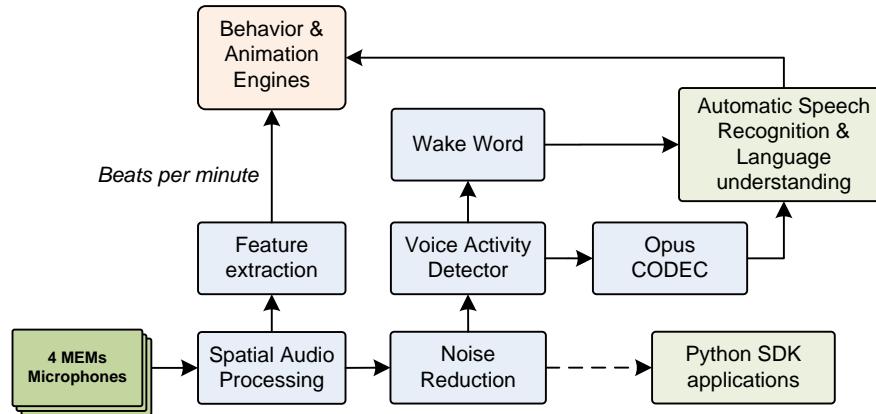


Figure 28: The audio input functional block diagram

- 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:

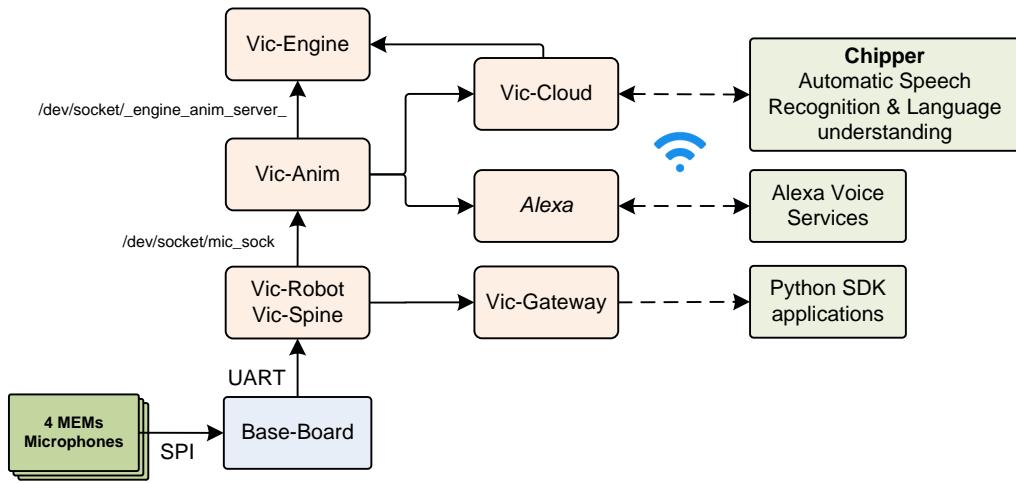


Figure 29: The audio input architecture

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.

28.1. 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.

28.2. 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.

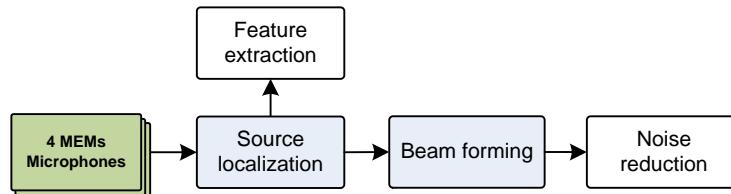


Figure 30: Typical spatial audio processing flow

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

28.3. NOISE REDUCTION

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

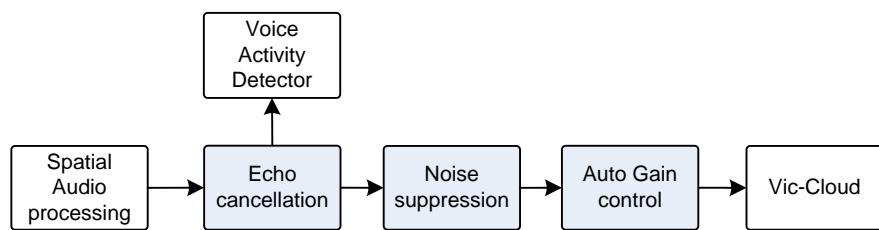


Figure 31: Typical audio noise reduction flow

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).

28.4. 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. If there was an intent found (and control is not reserved), the intent is mapped to a local behaviour to be carried out. This is described in a later section.

28.4.1 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
<code>alexa_thf</code>	array of TBD	The
<code>hey_vector_thf</code>	array of TBD	

Table 20: The `micTriggerConfig` JSON structure

Each TBD structure has the following fields:

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

Table 21: The TBD JSON structure

28.5. CLOUD SPEECH RECOGNITION

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

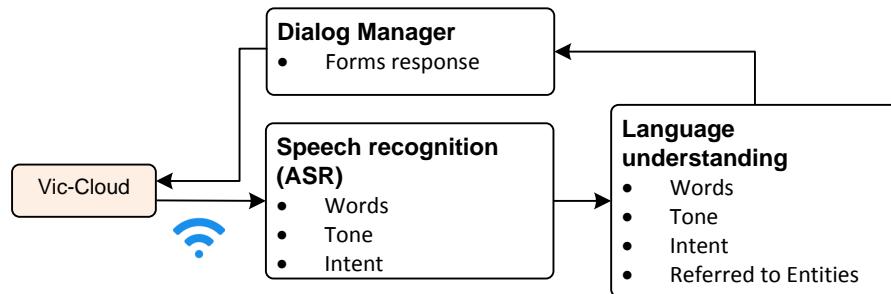


Figure 32: The speech recognition

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:

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

Table 22: The user_intent_map JSON structure

Each of the mapping entries has the following structure:

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

Table 23: The intent mapping JSON structure

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.

28.6. CONNECTIONS WITH VIC-GATEWAY AND SDK ACCESS

An application has access to the wake-word events and the received user intent events as they occur. When the "Hey, Vector" wake word is heard,

1. A WakeWordBegin event (part of the *WakeWord* event) is posted to Vic-Engine and Vic-Gateway. Vic-Gateway may forward the message on to a connected application.
2. A *StimulationInfo* event message, an emotion event "ReactToTriggerWord," is posted to Vic-Gateway for possible forwarding to a connected application.
3. A WakeWordEnd event (part of the *WakeWord* event) is sent (to Vic-Gateway for possible forwarding to a connected application) when the Vic-cloud has received a response back. If

control has not been reserved, and intent was received, the intent JSON data structure is included.

4. If there was no intent found, a *StimulationInfo* event message is posted (to Vic-Gateway), with an emotion event such as **NoValidVoiceIntent**
5. If there was an intent found (and control is reserved), a *UserIntent* event is posted to Vic-Gateway for possible forwarding to a connected application. In this case, the intent will not be carried out.

An external application can send an intent to Vector using the *App Intent* command (Chapter 14).

28.6.1 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 28.2 *Spatial audio processing*)
- 1600 audio samples; Note: this is 10x the chunk size of the internal processing size

29. REFERENCES AND RESOURCES

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

CHAPTER 11

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

30. 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.

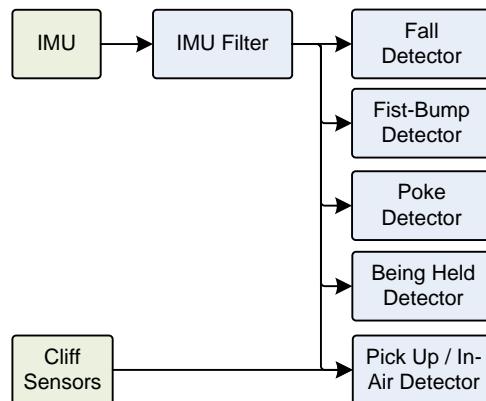


Figure 33: Motion sensing

30.1. 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.

Gyrosopes can measure rotations around those axes, but cannot measure linear motion along the axis. Gyrosopes 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.

The gyroscope is calibrated at start up.

30.2. 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.

30.3. 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.

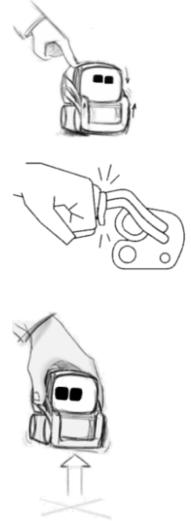
The IMU can be used to detect the angle of Vector's body. This is important, as the charging behaviour uses the tilt of the charging station ramp to know that it is in the right place.

30.4. 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

31. 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 exchanged, 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 12

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

32. 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:

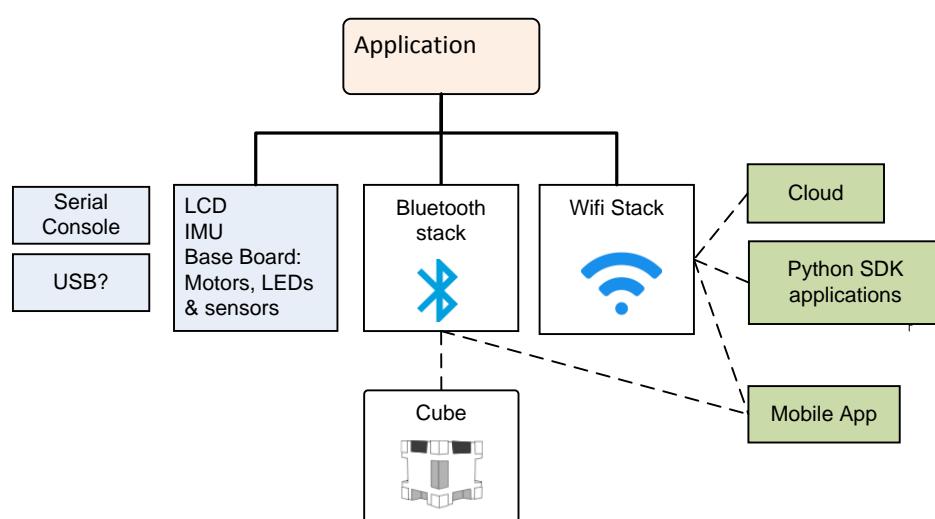


Figure 34: The overall communication infrastructure

33. INTERNAL COMMUNICATION WITH PERIPHERALS

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

33.1. 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/sec²⁸

33.1.1 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] AA₁₆ ‘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:

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.

33.1.2 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.

33.2. 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

33.3. 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 functions 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.

34. 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:

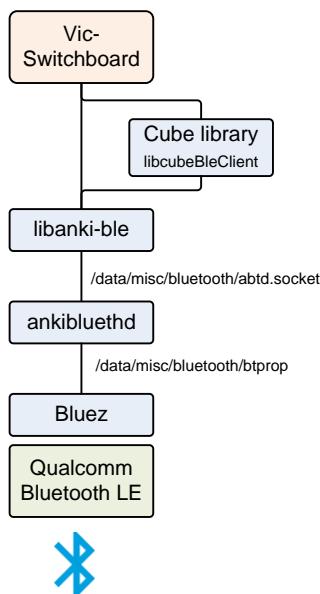


Figure 35: The Bluetooth LE stack

The elements of the Bluetooth LE stack include:

Element	Description & Notes
ankibluehd	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.

Table 25: Elements of the Bluetooth LE stack

35. 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 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:

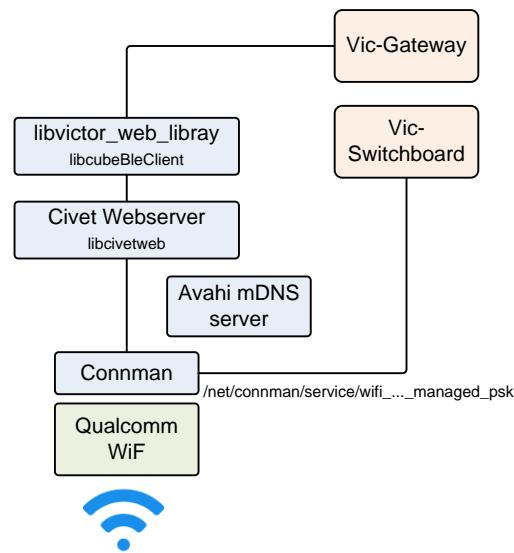


Figure 36: The WiFi stack

The elements of the stack include:³⁰

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

Table 26: Elements of the Bluetooth LE stack

35.1. FIREWALL

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

```
/etc/iptables/iptables.rules
/etc/iptables/ip6tables.rules
```

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

35.2. 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

36. 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

36.1. 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 14). 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/YIYQsX08OD4/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:

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

Table 27: Elements of a Vector certificate

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

File	Description
<i>AnkiRobotDeviceCert.pem</i>	The
<i>AnkiRobotDeviceKeys.pem</i>	The
<i>Info\${serialNum}.json</i>	A configuration file that
<i> \${serialNum}</i>	empty

Table 28: OEM cloud folder

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

Field	Type	Description
<i>CertDigest</i>	base64 string	
<i>CertSignature</i>	base64 string	
<i>CertSignatureAlgorithm</i>	string	The name of openSSL signature algorithm to use, “sha256WithRSAEncryption”
<i>CommonName</i>	string	‘vic:’ followed by the serial number. (This is also called the “thing id” in other structures.)
<i>KeysDigest</i>	base64 string	

Table 29: Cloud *Info\${serialNum}* structure

37. 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 13

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.

38. 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:

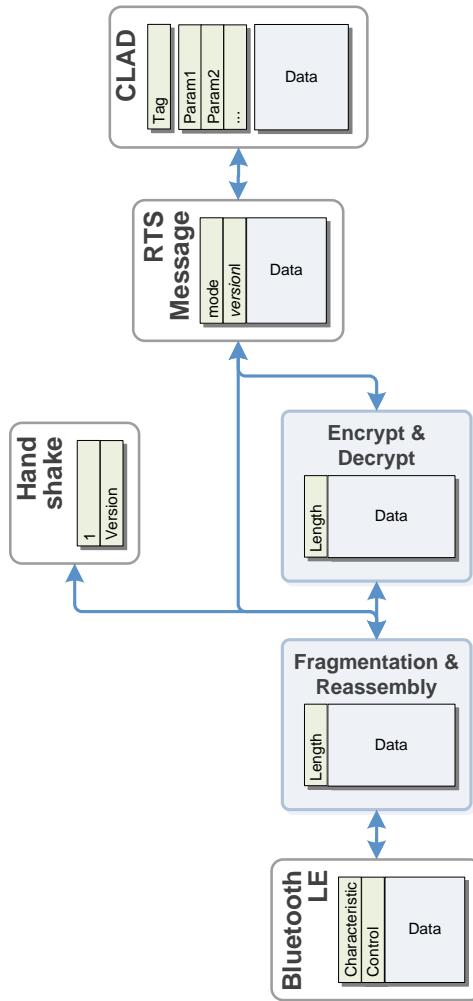


Figure 37: Overview of encryption and fragmentation stack

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,

38.1. 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:

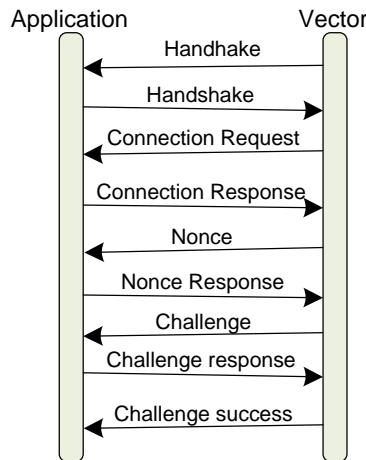


Figure 38: Sequence for initiating communication with Vector

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.

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

Table 30: Parameters for Handshake message

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 38.3.1 *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 38.3.1 *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.

38.2. 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 bit of the control byte.
5. Send the frame to Vector
6. If there are bytes remaining, repeat from step 2.

38.3. 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	Table 31: The encryption variables
<i>decryptionKey</i>	The key used to decrypt each message from to Vector.	
<i>decryptionNonce</i>	An extra bit that is added to each message. The initial nonce's to use are provided by Vector.	
<i>encryptionKey</i>	The key used to encrypt each message sent to Vector.	
<i>encryptionNonce</i>	An extra bit that is added to each message as it is encrypted. The initial nonce's to use are provided by Vector.	
<i>pinCode</i>	6 digits that are displayed by Vector during an initial pairing.	
<i>Vectors_publicKey</i>	The public key provided by Vector, used to create the encryption and decryption keys.	

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

- First time pairing, and
- Reconnection

38.3.1 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

38.3.2 Reconnecting

Reconnecting can reuse 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?}

38.3.3 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_xchacha20poly1305ietf_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_xchacha20poly1305ietf_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.

38.4. 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:

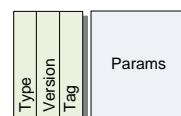


Figure 39: The format of an RTS frame

- 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.

38.5. FETCHING A LOG

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

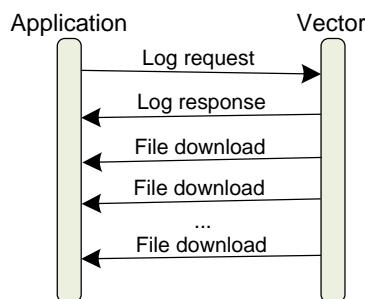


Figure 40: Sequence for initiating communication with Vector

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.

39. 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

	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

Table 32: Summary of the commands

39.1. APPLICATION CONNECTION ID

?

39.1.1 Request

The parameters of the request body are:

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

Table 33: Parameters for Application Connection Id request

39.1.2 Response

There is no response.

39.2. CANCEL PAIRING

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

39.2.1 Request

The command has no parameters.

39.2.2 Response

There is no response.

39.3. CHALLENGE

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

39.3.1 Request

The parameters of the request body are:

Offset	Size	Type	Parameter	Description
0	4	uint8_t	<i>value</i>	The challenge value

Table 34: Parameters for challenge request

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

39.3.2 Response

The parameters of the response body are:

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

Table 35: Parameters for challenge response

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

39.4. CHALLENGE SUCCESS

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

39.4.1 Request

The command has no parameters.

39.4.2 Response

There is no response.

39.5. CLOUD SESSION

This command is used to request a cloud session.

39.5.1 Command

The parameters of the request body are:

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

Table 36: Parameters for Cloud Session request

39.5.2 Response result

The parameters for the connection response message:

Offset	Size	Type	Parameter	Description
0	1	uint8_t	<i>success</i>	0 if failed, otherwise successful
1	1	uint8_t	<i>status</i>	See Table 38: Cloud status enumeration
2	1	uint16_t	<i>client token GUID length</i>	The number of bytes in the client token GUID; may be 0
	varies	uint8_t[]	<i>client token GUID</i>	The client token GUID. The client token GUID should be saved for future use.

Table 37: 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 38: Cloud status enumeration

³² <https://groups.google.com/forum/#!msg/anki-vector-rooting/YIYQsX08OD4/fvkAOZ91CgAJ>
<https://groups.google.com/forum/#!msg/anki-vector-rooting/XAaBE6e94ek/OdES50PaBQAJ>

5 authorized as secondary

6 reauthorization

39.6. CONNECT

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

39.6.1 Request

The parameters of the request body are:

Offset	Size	Type	Parameter	Description
0	32	uint8_t[32]	<i>publicKey</i>	The public key for the connection

Table 39: Parameters for Connection request

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

39.6.2 Response

The parameters for the connection response message:

Offset	Size	Type	Parameter	Description
0	1	uint8_t	<i>connectionType</i>	See Table 41: Connection types enumeration
1	32	uint8_t[32]	<i>publicKey</i>	The public key to use for the connection

Table 40: Parameters for Connection Response

The connection types are:

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

Table 41: Connection types enumeration

The application sends the response, with its *publicKey* (see section 38.3 *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.

39.7. 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.

39.7.1 Request

The command has no parameters.

39.7.2 Response

There is no response.

39.8. FILE DOWNLOAD

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

39.8.1 Request

There is no direct request.

39.8.2 Response

The parameters of the response body are:

Offset	Size	Type	Parameter	Description	Table 42: Parameters for File Download request
0	1	uint8_t	<i>status</i>		
1	4	uint32_t	<i>file id</i>		
5	4	uint32_t	<i>packet number</i>	The chunk within the download	
9	4	uint32_t	<i>packet total</i>	The total number of packets to be sent for this file download	
13	2	uint16_t	<i>length</i>	The number of bytes to follow (can be 0)	
varies		uint8_t[length]	<i>bytes</i>	The bytes of this file chunk	

39.9. LOG

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

39.9.1 Request

The parameters of the request body are:

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

Table 43: Parameters for Log request

Each filter entry has the following structure:

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

Table 44: Log filter

39.9.2 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:

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

Table 45: Parameters for Log Response

39.10. NONCE

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

39.10.1 Request

The parameters for the nonce request message:

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

Table 46: Parameters for Nonce request

39.10.2 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	<i>connection tag</i>	This is always 3

Table 47: 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.

39.11. OTA UPDATE

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

39.11.1 Request

The parameters of the request body are:

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

Table 48: Parameters for OTA request

39.11.2 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	<i>status</i>	See Table 50: OTA status enumeration
1	8	uint64_t	<i>current</i>	The number of bytes downloaded
9	8	uint64_t	<i>expected</i>	The number of bytes expected to be downloaded

Table 49: 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 347: OTA update-engine status codes for these update-engine status codes.

Table 50: OTA status enumeration

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

39.12. 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.

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

Table 51: Parameters for Response

39.13. 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.

39.13.1 Request

The parameters of the request body are:

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

Table 52: Parameters for the SDK proxy request

39.13.2 Response

The parameters for the response message:

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

Table 53: Parameters for the SDK proxy Response

39.14. 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.

39.14.1 Request

The parameters for the request message:

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

Table 54: Parameters for SSH request

Each authorization key has the following structure:

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

Table 55: SSH authorization key

39.14.2 Response

The response has no parameters.

39.15. STATUS

This command is used to request basic info from Vector.

39.15.1 Request

The request has no parameters.

39.15.2 Response

The parameters for the response message:

Offset	Size	Type	Parameter	Description	Table 56: Parameters for Status Response
0	1	uint8_t	<i>SSID length</i>	The number of bytes in the SSID string; may be 0	
2	varies	uint8_t[<i>SSID length</i>]	<i>SSID</i>	The WiFi SSID (hex string).	
1	uint8_t		<i>WiFi state</i>	See Table 57: WiFi state enumeration	
1	uint8_t		<i>access point</i>	0 not acting as an access point, otherwise acting as an access point	
1	uint8_t		<i>Bluetooth LE state</i>	0 if the Bluetooth	
1	uint8_t		<i>Battery state</i>		
1	uint8_t		<i>version length</i>	The number of bytes in the version string; may be 0 version >= 2	
varies	uint8_t [<i>version length</i>]		<i>version</i>	The version string; version >= 2	
1	uint8_t		<i>ESN length</i>	The number of bytes in the ESN string; may be 0 version >= 4	
varies	uint8_t[<i>ESN length</i>]		<i>ESN</i>	The <i>electronic serial number</i> string; version >= 4	
1	uint8_t		<i>OTA in progress</i>	0 over the air update not in progress, otherwise in process of over the air update; version >= 2	
1	uint8_t		<i>has owner</i>	0 does not have owner, otherwise has owner; version >= 3	
1	uint8_t		<i>cloud authorized</i>	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:

Index	Meaning	Table 57: WiFi state enumeration
0	Unknown	
1	Online	
2	Connected	
3	Disconnected	

39.16. 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 39.5 *Cloud session*).

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

39.16.1 Request

The parameters of the request body are:

Offset	Size	Type	Parameter	Description
0	1	uint8_t	<i>enable</i>	0 to disable the WiFi access point, 1 to enable it

Table 58: Parameters for WiFi Access Point request

39.16.2 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	<i>enabled</i>	0 if the WiFi access point is disabled, otherwise enabled
1	1	uint8_t	<i>SSID length</i>	The number of bytes in the SSID string; may be 0
2	varies	uint8_t[SSID length]	<i>SSID</i>	The WiFi SSID (hex string)
1	uint8_t		<i>password length</i>	The number of bytes in the password string; may be 0
varies	uint8_t [password length]		<i>password</i>	The WiFi password

Table 59: Parameters for WiFi Access Point Response

39.17. WIFI CONNECT

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

39.17.1 Request

The parameters for the request message:

Offset	Size	Type	Parameter	Description
0	1	uint8_t	<i>SSID length</i>	The number of bytes in the SSID string; may be 0
1	varies	uint8_t[<i>SSID length</i>]	<i>SSID</i>	The WiFi SSID (hex string)
1	uint8_t		<i>password length</i>	The number of bytes in the password string; may be 0
varies	uint8_t [<i>password length</i>]		<i>password</i>	The WiFi password
1	uint8_t		<i>timeout</i>	How long to given the connect attempt to succeed.
1	uint8_t		<i>auth type</i>	The type of authentication to employ; see <i>Table 61: WiFi authentication types enumeration</i>
1	uint8_t		<i>hidden</i>	0 the access point is not hidden; 1 it is hidden

Table 60: Parameters for WiFi Connect request

The WiFi authentication types are:

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

Table 61: WiFi authentication types enumeration

39.17.2 Response

The parameters for the response message:

Offset	Size	Type	Parameter	Description
0	1	uint8_t	<i>SSID length</i>	The length of the SSID that was deleted; may be 0
1	varies	uint8_t[<i>SSID length</i>]	<i>SSID</i>	The SSID (hex string) that was deleted
1	uint8_t		<i>WiFi state</i>	See <i>Table 57: WiFi state enumeration</i>
1	uint8_t		<i>connect result</i>	version >= 3

Table 62: Parameters for WiFi Connect command

39.18. WIFI FORGET

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

39.18.1 Request

The parameters for the request message:

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

Table 63: Parameters for WiFi Forget request

39.18.2 Response

The parameters for the response message:

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

Table 64: Parameters for WiFi Forget response

39.19. WIFI IP ADDRESS

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

39.19.1 Request

The request has no parameters

39.19.2 Response

The parameters for the response message:

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

Table 65: Parameters for WiFi IP Address response

39.20. WIFI SCAN

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

39.20.1 Request

The command has no parameters.

39.20.2 Response

The parameters for the response message:

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

Table 66: Parameters for WiFi scan response

Each access point has the following structure:

Offset	Size	Type	Parameter	Description
0	1	uint8_t	<i>auth type</i>	The type of authentication to employ; see <i>Table 61: WiFi authentication types enumeration</i>
1	1	uint8_t	<i>signal strength</i>	The number of bars, 0..4
2	1	uint8_t	<i>SSID length</i>	The length of the SSID string
3	varies	uint8_t[SSID length]	<i>SSID</i>	The SSID (hex string)
	1	uint8_t	<i>hidden</i>	0 not hidden, 1 hidden; version >= 2
	1	uint8_t	<i>provisioned</i>	0 not provisioned, 1 provisioned; version>= 3

Table 67: Parameters access point structure

CHAPTER 14

The HTTPS based API

This chapter describes the communication with Vector via the local HTTPS. This is intended to be supplemental information not available at:

<https://developer.anki.com/vector/docs/proto.html>

The Anki documentation includes descriptions of the following types:

- `ActionResult`
- `ResponseStatus`

Note: the information in this chapter comes from the protobuf specification files in the python SDK, from the SDK itself, and some analysis of the mobile application.

40. OVERVIEW OF THE SDK HTTPS API

The descriptions below³³ give the JSON keys, and their 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 implementation 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)

40.1. SDK MESSAGE GROUPINGS

The major groups of messages here are:

- Accessories and custom objects
- Actions and behaviors – setting the current priority and cancelling actions
- Alexa configuration – configuring Vector to use Alexa's services
- Audio – playing sounds on Vector, and submitting text to speech

³³ The protocol was specified in Google Protobuf.

- Battery – the current state of charge
- Connection – authenticating with the remote servers to allow access to Vector, connection management, event stream, end-point version info, and checking the connection with the cloud
- Cube – commands to manage and interact with the cube
- Display – display images on Vector’s LCD
- Faces (of people, not Vector’s face)
- Features and entitlements – the features that are enabled (or disabled)
- Image processing – Getting a video stream, and enabling (or disabling) video processing steps
- Interactions with objects (outside of the cube)
- JDocs, the JSON document storage interface
- Map and Navigation
- Miscellaneous items
- Motion Control
- Motion Sensing – how Vector senses that he is moving
- Onboarding
- Photos – commands to access (and delete) photographs and their thumbnails
- Settings and Preferences
- Software Updates, used to update Vector’s software – operating system, applications, assets, etc.

41. COMMON ELEMENTS

The enumerations and structures in this section are common to many commands.

41.1. ENUMERATIONS

41.1.1 ResultCode

The ResultCode enumeration has the following named values:

Name	Value	Description
ERROR_UPDATE_IN_PROGRESS	1	The settings could not be applied; there is already another update to the settings in process.
SETTINGS_ACCEPTED	0	The settings were successfully saved.

Table 68: ResultCode Enumeration

41.1.2 RobotStatus

The RobotStatus is a bit mask used to indicate the Vector is doing, and the status of his controls. 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	00001 ₁₆	This bit is set “if Vector is currently moving any of his motors (head, arm or wheels/treads).”
ROBOT_STATUS_ARE_MOTORS_MOVING	00001 ₁₆	
ROBOT_STATUS_IS_CARRYING_BLOCK	00002 ₁₆	This bit is set “if Vector is currently carrying a block.”
ROBOT_STATUS_IS_PICKING_OR_PLACING	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_DOCKING_TO_MARKER	00004 ₁₆	
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

Table 69: RobotStatus Enumeration

		position.” It is clear “if still trying to move there.”
<i>ROBOT_STATUS_CALM_POWER_MODE</i>	00400_{16}	This bit is set “if Vector is in calm power mode. Calm power mode is generally when Vector is sleeping or charging.”
<i>reserved</i>	00800_{16}	<i>This bit is not defined</i>
<i>ROBOT_STATUS_IS_ON_CHARGER</i>	01000_{16}	This bit is set “if Vector is currently on the charger.” Note: Vector may be on the charger without charging.
<i>ROBOT_STATUS_IS_CHARGING</i>	02000_{16}	This bit is set “if Vector is currently charging.”
<i>ROBOT_STATUS_CLIFF_DETECTED</i>	04000_{16}	This bit is set “if Vector detected a cliff using any of his four cliff sensors.”
<i>ROBOT_STATUS_ARE_WHEELS_MOVING</i>	08000_{16}	This bit is set “if Vector's wheels/treads are currently moving.”
<i>ROBOT_STATUS_IS_BEING_HELD</i>	10000_{16}	This bit is set “if Vector is being held.” Note: <i>ROBOT_STATUS_IS_PICKED_UP</i> will also be set when this bit is set. 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.
<i>ROBOT_STATUS_IS_MOTION_DETECTED</i> <i>ROBOT_STATUS_IS_ROBOT_MOVING</i>	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.

41.2. STRUCTURES

41.2.1 CladPoint

The CladPoint is used to represent a 2D rectilinear point on an image or in the 2D map. It has the following fields:

Field	Type	Units	Description
x	float	<i>pixels</i>	The x-coordinate of the point
y	float	<i>pixels</i>	The y-coordinate of the point

Table 70: CladPoint JSON structure

41.2.2 CladRect

The CladRect is used to represent a 2D rectilinear rectangle on an image. It has the following fields:

Field	Type	Units	Description
height	float	<i>pixels</i>	The height of the rectangle
width	float	<i>pixels</i>	The width of the rectangle
x_top_left	float	<i>pixels</i>	The x-coordinate of the top-left corner of the rectangle within the image.
y_top_left	float	<i>pixels</i>	The y-coordinate of the top-left corner of the rectangle within the image.

Table 71: CladRectangle JSON structure

41.2.3 PoseStruct

The PoseStruct is used to represent a 3D rectilinear point and orientation on the map. It has the following fields:

Field	Type	Units	Description
origin_id	uint32		Which coordinate frame this pose is in (0 for none or unknown).
q0	float		Part of the rotation quaternion
q1	float		
q2	float		Part of the rotation quaternion
q3	float		
x	float	<i>mm</i>	The translation x coordination
y	float	<i>mm</i>	The translation y coordination
z	float	<i>mm</i>	The translation z coordination

Table 72: PoseStruct JSON structure

42. ACCESSORIES AND CUSTOM OBJECTS

This section describes the objects that Vector can see and track in his map. Specialized accessories – the charger and cube – are broken out into their own sections.

See also *Cube* and *Interactions with Objects*

You too can create custom objects for Vector to... at least see and perceive. Maybe even love. There are four kinds of custom objects that you can define:

- A fixed, unmarked cube-shaped object. The object is in a fixed position and orientation, and it can't be observed (since it is unmarked). So there won't be any events related to this object. "This could be used to make Vector aware of objects and know to plot a path around them."
- A flat wall with only a front side,
- A cube, with the same marker on each side.
- A box with different markers on each side.

A note about object id's: The object id may change: "a cube disconnecting and reconnecting it's removed and then re-added to robot's internal world model which results in a new ID."

The client should employ a timer for each potential visual object. If there isn't an "object observed" event received in the time period, it should be assumed "that Vector can no longer see an object."

42.1. ENUMERATIONS

42.1.1 CustomObjectMarker

The `CustomObjectMarker` is used represent the marker symbol used. The symbols are predefined, with the images that Vector recognizes included in the SDK. The enumeration has the following named values:

Name	Value	Description	Table 73: <code>CustomObjectMarker</code> Enumeration
<code>CUSTOM_MARKER_UNKNOWN</code>	0		
<code>CUSTOM_MARKER_CIRCLES_2</code>	1		
<code>CUSTOM_MARKER_CIRCLES_3</code>	2		
<code>CUSTOM_MARKER_CIRCLES_4</code>	3		
<code>CUSTOM_MARKER_CIRCLES_5</code>	4		
<code>CUSTOM_MARKER_DIAMONDS_2</code>	5		
<code>CUSTOM_MARKER_DIAMONDS_3</code>	6		
<code>CUSTOM_MARKER_DIAMONDS_4</code>	7		
<code>CUSTOM_MARKER_DIAMONDS_5</code>	8		
<code>CUSTOM_MARKER_HEXAGONS_2</code>	9		
<code>CUSTOM_MARKER_HEXAGONS_3</code>	10		
<code>CUSTOM_MARKER_HEXAGONS_4</code>	11		

<i>CUSTOM_MARKER_HEXAGONS_5</i>	12
<i>CUSTOM_MARKER_TRIANGLES_2</i>	13
<i>CUSTOM_MARKER_TRIANGLES_3</i>	14
<i>CUSTOM_MARKER_TRIANGLES_4</i>	15
<i>CUSTOM_MARKER_TRIANGLES_5</i>	16
<i>CUSTOM_MARKER_COUNT</i>	16

42.1.2 CustomType

The CustomType is used to represent the identifier of object that a symbol is attached to. The enumeration has the following named values:

Name	Value	Description
<i>INVALID_CUSTOM_TYPE</i>	0	
<i>CUSTOM_TYPE_00</i>	1	
<i>CUSTOM_TYPE_01</i>	2	
<i>CUSTOM_TYPE_02</i>	3	
<i>CUSTOM_TYPE_03</i>	4	
<i>CUSTOM_TYPE_04</i>	5	
<i>CUSTOM_TYPE_05</i>	6	
<i>CUSTOM_TYPE_06</i>	7	
<i>CUSTOM_TYPE_07</i>	8	
<i>CUSTOM_TYPE_08</i>	9	
<i>CUSTOM_TYPE_09</i>	10	
<i>CUSTOM_TYPE_10</i>	11	
<i>CUSTOM_TYPE_11</i>	12	
<i>CUSTOM_TYPE_12</i>	13	
<i>CUSTOM_TYPE_13</i>	14	
<i>CUSTOM_TYPE_14</i>	15	
<i>CUSTOM_TYPE_15</i>	16	
<i>CUSTOM_TYPE_16</i>	17	
<i>CUSTOM_TYPE_17</i>	18	
<i>CUSTOM_TYPE_18</i>	18	
<i>CUSTOM_TYPE_19</i>	19	
<i>CUSTOM_TYPE_COUNT</i>	20	

Table 74: CustomType Enumeration

42.1.3 ObjectFamily

The ObjectFamily is a deprecated method used to represent the type of object that a symbol is attached to. ObjectType should be used instead, where possible. The enumeration has the following named values:

Name	Value	Description
INVALID_FAMILY	0	This value represents a kind of object that is not properly set.
UNKNOWN_FAMILY	1	This value is used when there is an object, but its kind is not known.
BLOCK	2	This is the identifier used for blocks/cubs other than the companion-cube
LIGHT_CUBE	3	This is the identifier used for the companion-cube
CHARGER	4	This is the identifier used for the home charging station.
CUSTOM_OBJECT	7	This is the identifier used for as custom object definition.
OBJECT_FAMILY_COUNT	7	

Table 75: ObjectType Enumeration

42.1.4 ObjectType

The ObjectType is used represent the type of object that a symbol is attached to. The enumeration has the following named values:

Name	Value	Description
INVALID_OBJECT	0	This value represents an object id used when there isn't an object associated.
UNKNOWN_OBJECT	1	This value is used when there is an object, but it is not recognized.
BLOCK_LIGHTCUBE1	2	This is the identifier used for the companion-cube
CHARGER_BASIC	6	This is the identifier used for the home charging station.
FIRST_CUSTOM_OBJECT_TYPE	15	The custom objects all have types greater than or equal to this.

Table 76: ObjectType Enumeration

42.2. EVENTS

42.2.1 ObjectEvent

The ObjectEvent event is sent (see *Event* message) when the state of an object has changed. The structure has one (and only one) of the following fields:

Field	Type	Description
<i>cube_connection_lost</i>	CubeConnectionLost	This event is sent when cube no longer is connected via Bluetooth LE.
<i>robot_observed_object</i>	RobotObservedObject	This even is sent the object is visually seen by Vector.
<i>object_available</i>	ObjectAvailable	This event is sent when cube a Bluetooth LE connection to the cube is established.
<i>object_connection_state</i>	ObjectConnectionState	The information about the Bluetooth LE identity of the cube, and whether is connected (or not).
<i>object_moved</i>	ObjectMoved	The object has changed position.
<i>object_stopped_moving</i>	ObjectStoppedMoving	The object had change position previously, but has now come to reset.
<i>object_tapped</i>	ObjectTapped	The cube was tapped.
<i>object_up_axis_changed</i>	ObjectUpAxisChanged	The object was rotated and has a new upward face.

Table 77: ObjectEvent JSON structure

42.2.2 ObjectAvailable

The ObjectAvailable event is sent (see *ObjectEvent*) when Vector has received Bluetooth LE advertisements from the object (cube).

See also *CubeConnectionLost*, when the cube is disconnected.

This event structure has the following fields:

Field	Type	Units	Description
<i>factory_id</i>	string		The identifier for the cube. This is built into the cube.

Table 78: ObjectAvailable JSON structure

42.2.3 ObjectConnectedState

The ObjectConnectedState event is to “indicate that a cube has connected or disconnected to the robot. This message will be sent for any connects or disconnects regardless of whether it originated from us or underlying robot behavior.”

See also *CubeConnectionLost*.

This event structure has the following fields:

Field	Type	Units	Description
<i>connected</i>	bool		True if Vector has a Bluetooth LE connection with the Cube.
<i>factory_id</i>	string		The identifier for the cube. This is built into the cube.
<i>object_id</i>	uint32		The identifier of the object that Vector is (or was) connected to.
<i>object_type</i>	ObjectType		The type of object referred to.

Table 79:
ObjectConnectedState
JSON structure

42.2.4 ObjectMoved

The ObjectMoved event is sent (see *ObjectEvent*) when an object has changed its position. The structure has the following fields:

Field	Type	Units	Description
<i>object_id</i>	uint32		The identifier of the object that moved.
<i>timestamp</i>	uint32		The time that the event occurred on. The format is milliseconds since Vector’s epoch.

Table 80: *ObjectMoved*
JSON structure

42.2.5 ObjectStoppedMoving

The ObjectStoppedMoving event is sent (see *ObjectEvent*) when an object previously identified as moving has come to rest. The structure has the following fields:

Field	Type	Units	Description
<i>object_id</i>	uint32		The identifier of the object that was moving.
<i>timestamp</i>	uint32		The time that the event occurred on. The format is milliseconds since Vector’s epoch.

Table 81:
ObjectStoppedMoving
JSON structure

42.2.6 ObjectTapped

The ObjectTapped event is sent (see *ObjectEvent*) when an object has received a finger-tap. The structure has the following fields:

Field	Type	Units	Description
<i>object_id</i>	uint32		The identifier of the object tapped.
<i>timestamp</i>	uint32		The time that the event occurred on. The format is milliseconds since Vector's epoch.

Table 82: ObjectTapped JSON structure

42.2.7 ObjectUpAxisChanged

The ObjectUpAxis event is sent (see *ObjectEvent*) if the orientation of the object has significantly changed, leaving it with a new face upward. The structure has the following fields:

Field	Type	Units	Description
<i>object_id</i>	uint32		The identifier of the object whose axis has changed.
<i>timestamp</i>	uint32		The time that the event occurred on. The format is milliseconds since Vector's epoch.
<i>up_axis</i>	UpAxis		The orientation of object, represented as which axis is pointing upwards

Table 83: ObjectUpAxis JSON structure

The UpAxis is used represent the orientation of an object. The enumeration has the following named values:

Name	Value	Description
<i>INVALID_AXIS</i>	0	The orientation of the object is not known.
<i>X_NEGATIVE</i>	1	The positive direction along the body's x-axis is upward.
<i>X_POSITIVE</i>	2	The negative direction along the body's x-axis is upward.
<i>Y_NEGATIVE</i>	3	The positive direction along the body's y-axis is upward.
<i>Y_POSITIVE</i>	4	The negative direction along the body's y-axis is upward.
<i>Z_NEGATIVE</i>	5	The positive direction along the body's z-axis is upward.
<i>Z_POSITIVE</i>	6	The negative direction along the body's z-axis is upward.
<i>NUM_AXES</i>	7	

Table 84: UpAxis Enumeration

42.2.8 RobotObservedObject

The RobotObservedObject event is sent when “an object with [the] specified ID/Type was seen at a particular location in the image and the world.” This event structure has the following fields:

Field	Type	Units	Description	Table 85: RobotObservedObject JSON structure
<i>img_rect</i>	CladRect		The position of the object within the vision image.	
<i>is_active</i>	uint32			
<i>object_family</i>	ObjectFamily		<i>Deprecated.</i> “Use ObjectType instead to reason about groupings of objects.”	
<i>object_id</i>	int32		The identifier of the object that has been seen. Note that this is signed (int32 instead of uint32) for internal compatibility reasons.	
<i>object_type</i>	ObjectType		The type of object referred to.	
<i>pose</i>	PoseStruct		The observed pose of this object. <i>Optional.</i>	
<i>timestamp</i>	uint32		The time that the object was most recently observed. The format is milliseconds since Vector’s epoch.	
<i>top_face_orientation_rad</i>	float	radians	“Angular distance from the current reported up axis. “ “absolute orientation of top face, iff isActive==true”	

42.3. CREATE FIXED CUSTOM OBJECT

This command “creates a permanent custom [cube-shaped] object instance in the robot's world” except this object has “no markers associated with it.” The object “will remain in the specified pose as an obstacle forever (or until deleted).” The object can't be observed, and won't create any events related to being observed. The fixed, custom object can “be used to make Vector aware of objects and know to plot a path around them.”

Post: “/v1/create_fixed_custom_object”

42.3.1 Request

The CreateFixedCustomObjectRequest structure has the following fields:

Field	Type	Units	Description
<i>pose</i>	PoseStruct		The position and orientation of this object.
<i>x_size_mm</i>	float	mm	The size of the object that the marker symbol is on, along the x-axis.
<i>y_size_mm</i>	float	mm	The size of the object that the marker symbol is on, along the y-axis.
<i>z_size_mm</i>	float	mm	The size of the object that the marker symbol is on, along the z-axis.

Table 86:
CreateFixedCustomObjectRequest JSON structure

42.3.2 Response

The CreateFixedCustomObjectResponse structure has the following fields:

Field	Type	Description
<i>object_id</i>	uint32	The object identifier assigned to this object.
<i>status</i>	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 87:
CreateFixedCustomObjectResponse JSON structure

42.4. DEFINE CUSTOM OBJECT

“Creates a custom object with distinct custom marker(s)” on one or more its faces. This can create a wall, a box, a cube (similar to a box, but each side is the same size as every other, and has the same marker). Once the object has been created, “the robot will now detect the markers associated with this object and send an RobotObservedObject message when they are seen. The markers must be placed in the center of their respective sides.”

Note: “No instances of this object are added to the world until they have been seen.”

See also *Create Fixed Custom Object*, *Delete Custom Objects*

Post: “/v1/define_custom_object”

42.4.1 Request

The DefineCustomObjectRequest structure has the following fields:

Field	Type	Units	Description	Table 88: DefineCustomObjectReq uest JSON structure
<i>custom_type</i>	CustomType		The object type to be assigned to this object.	
<i>is_unique</i>	bool		If true, “there is guaranteed to be no more than one object of this type present in the world at a time.”	
<i>custom_box</i>	CustomBoxDefinition		The definition of a box with different markers on each side.	
<i>custom_cube</i>	CustomCubeDefinition		The definition of a cube, with the same marker on each side.	
<i>custom_wall</i>	CustomWallDefinition		The definition of a flat wall with only a front side.	

Note: only one of “*custom_box*,” “*custom_cube*,” or “*custom_wall*” can be used in the request.

The CustomBoxDefinition “defines a custom object of the given size with the given markers centered on each side.” The structure has the following fields:

Field	Type	Units	Description
<i>marker_back</i>	CustomObjectMarker		The marker symbol used on the back surface of the box. This marker must be unique (not used by any of the other side’s on this box or in any other shape).
<i>marker_bottom</i>	CustomObjectMarker		The marker symbol used on the bottom surface of the box. This marker must be unique (not used by any of the other side’s on this box or in any other shape).
<i>marker_front</i>	CustomObjectMarker		The marker symbol used on the front surface of the box. This marker must be unique (not used by any of the other side’s on this box or in any other shape).
<i>marker_left</i>	CustomObjectMarker		The marker symbol used on the left-hand side of the box. This marker must be unique (not used by any of the other side’s on this box or in any other shape).
<i>marker_right</i>	CustomObjectMarker		The marker symbol used on the right-hand side of the box. This marker must be unique (not used by any of the other side’s on this box or in any other shape).
<i>marker_top</i>	CustomObjectMarker		The marker symbol used on the top surface of the box. This marker must be unique (not used by any of the other side’s on this box or in any other shape).
<i>marker_height_mm</i>	float	mm	The height of the marker symbol.
<i>marker_width_mm</i>	float	mm	The width of the marker symbol.
<i>x_size_mm</i>	float	mm	The size of the object, along the x-axis, that the marker symbol is on.
<i>y_size_mm</i>	float	mm	The size of the object, along the y-axis, that the marker symbol is on.
<i>z_size_mm</i>	float	mm	The width of the object, along the z-axis, that the marker symbol is on.

The CustomCubeDefinition “defines a custom cube of the given size.” The structure has the following fields:

Field	Type	Units	Description
<i>marker</i>	CustomObjectMarker		The marker symbol used on all of the cube surfaces; “the same marker [must] be centered on all faces.”
<i>marker_height_mm</i>	float	mm	The height of the marker symbol
<i>marker_width_mm</i>	float	mm	The width of the marker symbol
<i>size_mm</i>	float	mm	The height, width, and depth of the object that the marker symbol is on.

Table 89:
CustomBoxDefinition
JSON structure

The `CustomWallDefinition` “defines a custom wall of the given height and width... The wall's thickness is assumed to be 1cm (and thus there are no markers on its left, right, top, or bottom).” The structure has the following fields:

Field	Type	Units	Description
<code>marker</code>		<code>CustomObjectMarker</code>	The marker symbol used on the wall surfaces; “the same marker centered on both sides (front and back)”
<code>marker_height_mm</code>	float	<i>mm</i>	The height of the marker symbol
<code>marker_width_mm</code>	float	<i>mm</i>	The width of the marker symbol
<code>height_mm</code>	float	<i>mm</i>	The height of the object that the marker symbol is on.
<code>width_mm</code>	float	<i>mm</i>	The width of the object that the marker symbol is on.

Table 91:
`CustomWallDefinition`
JSON structure

42.4.2 Response

The `DefineCustomObjectResponse` type has the following fields:

Field	Type	Description
<code>status</code>	<code>ResponseStatus</code>	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.
<code>success</code>	bool	True if the thumbnail was successfully retrieved; otherwise there was an error.

Table 92:
`DefineCustomObjectResponse`
JSON structure

42.5. DELETE CUSTOM OBJECTS

This command “causes the robot to forget about custom objects it currently knows about.” All custom objects that match the given pattern are removed.

Post: “/v1/delete_custom_objects”

42.5.1 Request

The DeleteCustomObjectsRequest type has the following fields:

Field	Type	Description
<i>mode</i>	CustomObjectDeletionMode	The kind of custom objects to remove.

Table 93:
DeleteCustomObjectsRequest JSON structure

The CustomObjectDeletionMode is used to specify which kinds of custom objects should be deleted from the internal database. The enumeration has the following named values:

Name	Value	Description
<i>DELETION_MASK_UNKNOWN</i>	0	
<i>DELETION_MASK_FIXED_CUSTOM_OBJECTS</i>	1	Delete the custom objects that are “fixed” (don't have any marker symbols).
<i>DELETION_MASK_CUSTOM_MARKER_OBJECTS</i>	2	Delete the objects with marker symbols.
<i>DELETION_MASK_ARCHETYPES</i>	3	

Table 94:
CustomObjectDeletionMode Enumeration

42.5.2 Response

The DeleteCustomObjectsResponse type has the following fields:

Field	Type	Description
<i>status</i>	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 95:
DeleteCustomObjectsResponse JSON structure

43. ACTIONS AND BEHAVIOUR

“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.”

See also Interactions, which covers actions/behaviors that involve interacting with objects and faces

Actions may have tags, and have result code. Behaviors do not.

43.1. ENUMERATIONS

43.1.1 ActionResults

These are already described elsewhere

43.1.2 ActionTagConstants

This is the range of numbers in which we can assign an identifier for the action so that we can cancel it later.

Name	Value	Description
<code>INVALID_SDK_TAG</code>	0	
<code>FIRST_SDK_TAG</code>	2000001	An assigned action tag must be equal to or greater than this value.
<code>LAST_SDK_TAG</code>	3000000	An assigned action tag must be less than or equal to this value.

43.1.3 BehaviorResults

The BehaviorResults is used TBD. The enumeration has the following named values:

Name	Value	Description
<code>BEHAVIOR_INVALID_STATE</code>	0	
<code>BEHAVIOR_COMPLETE_STATE</code>	1	
<code>BEHAVIOR_WONT_ACTIVATE_STATE</code>	2	

43.2. EVENTS

43.2.1 StimulationInfo

The StimulationInfo event is used TBD. The structure has the following fields:

Field	Type	Units	Description	Table 98: StimulationInfo JSON structure
accel	float	mm/sec ²	The acceleration at the time of the stimulation.	
emotion_events	string[]		The list of event names related to the emotion. <i>Optional.</i>	
max_value	float		The minimum stimulation value. Typically 1	
min_value	float		The maximum stimulation value. Typically 0	
value	float		The stimulation value after applying the events.	
value_before_event	float		The stimulation value before the event(s). “matches value if there were no emotion events”	
velocity	float	mm/sec	The speed at the time of the stimulation.	

43.3. CANCEL ACTION BY ID TAG

Cancel “a previously-requested action.”

Post: “/v1/cancel_action_by_id_tag”

43.3.1 Request

The CancelActionByIdTagRequest structure has the following fields:

Field	Type	Description	Table 99: CancelActionByIdTagRe quest JSON structure
id_tag	uint32	“Use the id_tag provided to the action request”	

43.3.2 Response

The CancelActionByIdTagResponse type has the following fields:

Field	Type	Description	Table 100: CancelActionByIdTagRe sponse JSON structure
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.	

43.4. LOOK AROUND IN PLACE

This has Vector turn around (in place) and see what is around him. See also face observed, RobotObservedObject

Post: “/v1/look_around_in_place”

43.4.1 Request

The LookAroundInPlaceRequest structure has no fields.

43.4.2 Response

The LookAroundInPlaceResponse structure has the following fields:

Field	Type	Description
<i>result</i>	BehaviorResults	
<i>status</i>	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 101:
LookAroundInPlaceResponse JSON structure

44. ALEXA

44.1. ENUMERATIONS

44.1.1 AlexaAuthState

The AlexaAuthState is used represent how far in the Alexa Voice Services authorization process Vector is. The enumeration has the following named values:

Name	Value	Description	<i>Table 102: AlexaAuthState Enumeration</i>
ALEXA_AUTH_INVALID	0	“Invalid/error/versioning issue”	
ALEXA_AUTH_UNINITIALIZED	1	“Not opted in, or opt-in attempted but failed”	
ALEXA_AUTH_REQUESTING_AUTH	2	“Opted in, and attempting to authorize”	
ALEXA_AUTH_WAITING_FOR_CODE	3	“Opted in, and waiting on the user to enter a code”	
ALEXA_AUTH_AUTHORIZED	4	“Opted in, and authorized / in use”	

44.2. EVENTS

44.2.1 AlexaAuthEvent

The AlexaAuthEvent is used to post updates to SDK application when the authorization with Alexa Voice Services change. The structure has the following fields:

Field	Type	Description	<i>Table 103: AlexaAuthEvent JSON structure</i>
auth_state	AlexaAuthState		
extra	string		

44.3. ALEXA AUTH STATE

This is used to find out whether Vector has been authenticated and authorized to use Alexa Voice Services.

Post: “/v1/alexa_auth_state”

44.3.1 Request

The AlexaAuthStateRequest structure has no fields.

44.3.2 Response

The AlexaAuthStateResponse structure has the following fields:

Field	Type	Description
<i>auth_state</i>	AlexaAuthState	
<i>extra</i>	string	
<i>status</i>	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 104:
AlexaAuthStateResponse
JSON structure

44.4. ALEXA OPT IN

Post: “/v1/alexa_opt_in”

44.4.1 Request

The AlexaOptInRequest structure has the following fields:

Field	Type	Description
<i>opt_in</i>	bool	True, if Vector should employ Alexa Voice services; otherwise Vector should not.

Table 105:
AlexaOptInRequest
JSON structure

44.4.2 Response

The AlexaOptInResponse structure has the following fields:

Field	Type	Description
<i>status</i>	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 106:
AlexaOptInResponse
JSON structure

45. ATTENTION TRANSFER

45.1. EVENTS

45.1.1 AttentionTransfer

This event is sent when TBD. The AttentionTransfer structure has the following fields:

Field	Type	Description
<i>reason</i>	AttentionTransferReason	The reason that the attention was changed.
<i>seconds_ago</i>	float	How long ago the attention was changed.

Table 107:
AttentionTransfer
JSON structure

The AttentionTransferReason is used to represent why the attention was transferred. The enumeration has the following named values:

Name	Value	Description
<i>Invalid</i>	0	
<i>NoCloudConnection</i>	1	
<i>NoWifi</i>	2	
<i>UnmatchedIntent</i>	3	

Table 108:
AttentionTransferReason
Enumeration

45.2. GET LATEST ATTENTION TRANSFER

Part of the behaviour component

Post: “/v1/get_latest_attention_transfer”

45.2.1 Request

The GetLatestAttentionTransferRequest has no fields.

45.2.2 Response

The GetLatestAttentionTransferResponse has the following fields:

Field	Type	Description
<i>latest_attention_transfer</i>	LatestAttentionTransfer	
<i>status</i>	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 109:
GetLatestAttentionTransferResponse
JSON structure

The LatestAttentionTransfer structure has the following fields:

Field	Type	Description
<i>attention_transfer</i>	AttentionTransfer	When and why the attention was changed.

Table 110:
LatestAttentionTransfer
JSON structure

46. AUDIO

This section describes events and commands related to Vectors audio input and output.

46.1. ENUMERATIONS

46.1.1 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
<code>VOLUME_LOW</code>	0	
<code>VOLUME_MEDIUM_LOW</code>	1	
<code>VOLUME_MEDIUM</code>	2	
<code>VOLUME_MEDIUM_HIGH</code>	3	
<code>VOLUME_HIGH</code>	4	

Table 111:
`MasterVolumeLevel`
Enumeration

46.1.2 UtteranceState

The `UtteranceState` is used to represent the state of audio playback by Vector, including text to speech. It is used in the `SayTextResponse`. The enumeration has the following named values:

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

Table 112:
`UtteranceState`
Enumeration

46.2. EVENTS

The following events are sent in the *Event* message. When a person speaks the wake word, the *WakeWordBegin* event will be sent, followed by the *WakeWordEnd* event and possibly a *UserIntent* event.

46.2.1 UserIntent

The *UserIntent* event is sent by Vector when an intent is received (from the cloud), after a person has said the wake word and spoken. The *UserIntent* structure has the following fields:

Field	Type	Description
<i>intent_id</i> ³⁴	uint32	The identifier for the intent. See Appendix I, <i>Table 362: The “Hey Vector” phrases</i> for an enumeration.
<i>json_data</i>	string	The parameters as a JSON formatted string. This may be empty if there is not additional information.

Table 113: *UserIntent* JSON structure

46.2.2 WakeWord

This event is sent when 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
<i>wake_word_begin</i>	<i>WakeWordBegin</i>	This is sent when the wake word is heard. The structure has no contents.
<i>wake_word_end</i>	<i>WakeWordEnd</i>	This is sent when the response (and potential intent) is received from the cloud. This is sent before the <i>UserIntent</i> event (if any).

Table 114: *WakeWord* JSON structure

The *WakeWordEnd* structure has the following fields:

Field	Type	Description
<i>intent_heard</i>	bool	True if a sentence was recognized with an associated intent; false otherwise.
<i>intent_json</i>	string	The intent and parameters as a JSON formatted string. This is empty if an intent was not heard (<i>intent_heard</i> will be false), or if the client does not have control. In the later case, a <i>UserIntent</i> event with the intent JSON data will be sent.

Table 115: *WakeWordEnd* JSON structure

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

46.3. APP INTENT

This command allows the mobile application or SDK application to send an intent to Vector.

See also *UserIntent*, and *WakeWord*

Post: “/v1/app_intent”

46.3.1 Request

The AppIntentRequest structure has the following fields:

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

Table 116: 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

46.3.2 Response

The AppIntentResponse has the following fields:

Field	Type	Description
<i>status</i>	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 117:
AppIntentResponse
JSON structure

46.4. MASTER VOLUME

This command is used to set the volume of Vector's audio playback and sound effects.

46.4.1 Request

The *MasterVolumeResponse* has the following fields:

Field	Type	Description
<i>volume_level</i>	MasterVolumeLevel	This is used to set the volume of Vector's audio playback.

Table 118:
MasterVolumeRequest
JSON structure

46.4.2 Response

The *MasterVolumeResponse* has the following fields:

Field	Type	Description
<i>status</i>	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 119:
MasterVolumeResponse
JSON structure

46.5. SAY TEXT

This command is used to request the state of Vector speak the given text.

Post: “/v1/say_text”

46.5.1 Request

The SayTextRequest structure has the following fields:

Field	Type	Units	Description
<i>duration_scalar</i>	float	<i>ratio</i>	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
<i>text</i>	string		The text (the words) that Vector should say.
<i>use_vector_voice</i>	bool		True if the text should be spoken in “Vector’s robot voice; otherwise, he uses a generic human male voice.”

Table 120:
SayTextRequest JSON structure

46.5.2 Response

The SayTextResponse structure has the following fields:

Field	Type	Description
<i>state</i>	UtteranceState	Where in the speaking process Vector is currently.
<i>status</i>	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 121:
SayTextResponse JSON structure

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

47. BATTERY

(See also *RobotStatus* for a flag indicating that vector is charging.

See the motion control for actions to drive onto the charger.

47.1. ENUMERATIONS

The *BatteryLevel* enumeration is located in Chapter 8, *Power Management*, *Table 18: BatteryLevel codes as they apply to Vector*

47.2. 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”

47.2.1 Request

No parameters

47.2.2 Response

The *BatteryStateResponse* structure has the following fields:

Table 122:
BatteryStateResponse
JSON structure

Field	Type	Units	Description
<i>battery_level</i>	BatteryLevel		The interpretation of the battery level.
<i>battery_volts</i>	float	volts	The battery voltage.
<i>cube_battery</i>	CubeBatteryLevel		The status of the companion Cube’s battery.
<i>is_on_charger_platform</i>	bool		True if Vector is on his “home,” aka charger.
<i>is_charging</i>	bool		True if Vector is charging, false otherwise.
<i>status</i>	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.
<i>suggested_charger_sec</i>	float	seconds	Suggested amount of time to charge.

48. CONNECTION

48.1. ENUMERATIONS

48.1.1 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	Table 123: ConnectionCode Enumeration
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	

48.2. EVENTS

48.2.1 ConnectionResponse

The ConnectionResponse structure has the following fields:

Field	Type	Description	Table 124: ConnectionResponse JSON structure
<i>is_primary</i>	bool		
<i>status</i>	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.	

48.2.2 Event

The Event structure is to deliver messages that some event has occurred. It is received in periodic response to the part of the *Event Stream* command. All the events are carried in this one has one (and only) of the following fields:

Field	Type	Description	Table 125: Event JSON structure
<i>alexa_auth_event</i>	AlexaAuthEvent		
<i>attention_transfer</i>	AttentionTransfer		
<i>check_update_status_response</i>	CheckUpdateStatusResponse		
<i>connection_response</i>	ConnectionResponse		
<i>cube_battery</i>	CubeBattery		
<i>jdocs_changed</i>	JdocsChanged		
<i>keep_alive</i>	KeepAlivePing	“Used by Vector to verify the connection is still alive.”	
<i>mirror_mode_disabled</i>	MirrorModeDisabled		
<i>object_event</i>	ObjectEvent		
<i>onboarding</i>	Onboarding		
<i>photo_taken</i>	PhotoTaken		
<i>robot_state</i>	RobotState		
<i>robot_changed_observed_face_id</i>	RobotChangedObservedFaceID		
<i>robot_observed_face</i>	RobotObservedFace		
<i>stimulation_info</i>	StimulationInfo		
<i>time_stamped_status</i>	TimeStampedStatus		
<i>user_intent</i>	UserIntent		
<i>vision_modes_auto_disabled</i>	VisionModesAutoDisabled		
<i>wake_word</i>	WakeWord	This event is sent when the wake word has been heard.	

48.2.3 KeepAlivePing

This is “a null message used by streams to verify that the client is still connected.” This message has no fields.

48.2.4 TimeStampedStatus

The TimeStampedStatus structure has the following fields:

Field	Type	Description
<i>status</i>	Status	
<i>timestamp_utc</i>	uint32	The time that the status occurred on. The format is unix time: seconds since 1970, in UTC.

Table 126:
TimeStampedStatus
JSON structure

The Status structure has one (and only one) of the following fields:

Field	Type	Description
<i>face_enrollment_completed</i>	FaceEnrollmentComplete	
<i>feature_status</i>		
<i>meet_victor_face_scan_complete</i>	Meet Victor Face Scan Complete	
<i>meet_victor_face_scan_started</i>	Meet Victor Face Scan Started	

Table 127: *Status* JSON structure

48.3. CHECK CLOUD CONNECTION

This command is used to check the connection with the remote servers.

Post: “/v1/check_cloud_connection”

48.3.1 Request

The CheckCloudRequest has no fields.

48.3.2 Response

The CheckCloudResponse has the following fields:

Field	Type	Description
<i>code</i>	ConnectionCode	Whether the cloud is available, or the relevant connection error
<i>expected_packets</i>	int32	
<i>num_packets</i>	int32	
<i>status</i>	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.
<i>status_message</i>	string	

Table 128:
CheckCloudResponse
JSON structure

48.4. EVENT STREAM

This command is used to request a stream of events from Vector.

Post: “/v1/event_stream”

Get: “/v1/event_stream”

48.4.1 Request

The EventRequest has the following fields:

Field	Type	Description	Table 129: EventRequest JSON structure
<i>black_list</i>	FilterList	The list of events to not include. ?	
<i>connection_id</i>	string		
<i>white_list</i>	FilterList	The list of events to include.	

The FilterList structure has the following fields:

Field	Type	Description	Table 130: FilterList JSON structure
<i>list</i>	string[]	A list of events	

48.4.2 Response

The response is a stream of EventResponse structures. These have the following fields:

Field	Type	Description	Table 131: EventResponse JSON structure
<i>event</i>	Event	The event that occurred. This structure is described above in the subsection <i>Events</i>	
<i>status</i>	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.	

48.5. PROTOCOL VERSION

“Checks the supported protocol version by passing in the client version and minimum host version and receiving a response to see whether the versions are supported.”

Post: “/v1/protocol_version”

“The valid versions of the protocol. Protocol versions are updated when messages change significantly: new ones are added and removed, fields deprecated, etc. The goal is to support as many old versions as possible, only bumping the minimum when there is no way to handle a prior version.”

48.5.1 Request

The ProtocolVersionRequest has the following fields:

Field	Type	Description	Table 132: ProtocolVersionRequest JSON structure
<i>client_version</i>	int64	The version of the protocol that the client is using.	
<i>min_host_version</i>	int64	The minimum version level of the protocol that robot should support.	

48.5.2 Response

The ProtocolVersionResponse has the following fields:

Field	Type	Description	Table 133: ProtocolVersionResponse JSON structure
<i>host_version</i>	int64	The version of the protocol that the robot supports.	
<i>result</i>	Result	Whether or not the protocol version supported by the robot is compatible with the client. See below.	

The Result is used to indicate whether the client version is supported. The enumeration has the following named values:

Name	Value	Description	Table 134: Result Enumeration
<i>SUPPORTED</i>	1	The protocol supports the client version and the minimum host (robot) version of the protocol.	
<i>UNSUPPORTED</i>	0	The protocol is unable to support the client; either the client version is not supported, or the host is unable to support a compatible version of the protocol.	

48.6. SDK INITIALIZATION

“SDK-only message to pass version info for device OS, Python version, etc.”

Post: “/v1/sdk_initialization”

48.6.1 Request

The SDKInitializationRequest has the following fields:

Field	Type	Description	Table 135: SDKInitializationRequest JSON structure
<i>cpu_version</i>	string	The CPU model that the client (SDK) is using; <i>informational only</i> .	
<i>os_version</i>	string	The version of operating system that the client (SDK) is using; <i>informational only</i> .	
<i>python_implementation</i>	string		
<i>python_version</i>	string	The version of python that the client (SDK) is using. <i>Informational only</i> .	
<i>sdk_module_version</i>	string	The version of the SDK software that the client is using.	

48.6.2 Response

The SDKInitializationResponse type has the following fields:

Field	Type	Description	Table 136: SDKInitializationResponse JSON structure
<i>status</i>	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.	

48.7. USER AUTHENTICATION

This command is used to authenticate

Post: “/v1/user_authentication”

48.7.1 Request

The UserAuthenticationRequest has the following fields:

Field	Type	Description
<i>client_name</i>	bytes	
<i>user_session_id</i>	bytes	

Table 137:
UserAuthenticationRequest JSON structure

48.7.2 Response

The UserAuthenticationResponse has the following fields:

Field	Type	Description
<i>client_token_guid</i>	bytes	The token bytes to be included in subsequent HTTPS postings. This token should be saved for future use.
<i>code</i>	Code	The result of the authentication request
<i>status</i>	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 138:
UserAuthenticationResponse JSON structure

The Code enumeration is:

Name	Value	Description
<i>UNAUTHORIZED</i>	0	
<i>AUTHORIZED</i>	1	

Table 139: Code Enumeration

48.8. VERSION STATE

Retrieves Vector's version information.

Post: “/v1/version_state”

48.8.1 Request

The VersionStateRequest has no fields.

48.8.2 Response

The VersionStateResponse type has the following fields:

Table 140:
VersionStateResponse
JSON structure

Field	Type	Description
<i>engine_build_id</i>	string	The robot's software build identifier.
<i>status</i>	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.
<i>os_version</i>	string	The identifier of the robot's software version.

49. CUBE

This section describes the structures and commands to interact with the cube.

Comment: Many of the commands are specific to interacting with a cube, but appear to have been intended to be generalized to work with a wider range of objects

See also *Define Custom Object* for how to create custom box and cube objects.

The cube's unique identifier is called "factory_id" in these messages.

49.1. ENUMERATIONS

49.1.1 AlignmentType

The AlignmentType is used to indicate how Vector should align with the object. The enumeration has the following named values:

Name	Value	Description
ALIGNMENT_TYPE_UNKNOWN	0	
ALIGNMENT_TYPE_LIFT_FINGER	1	"Align the tips of the lift fingers with the target object"
ALIGNMENT_TYPE_LIFT_PLATE	2	"Align the flat part of the lift with the object (useful for getting the fingers in the cube's grooves)"
ALIGNMENT_TYPE_BODY	3	"Align the front of Vector's body (useful for when the lift is up)"
ALIGNMENT_TYPE_CUSTOM	4	"For use with distanceFromMarker parameter"

49.1.2 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

³⁵ The levels are from robot.py

49.2. EVENTS

49.2.1 CubeBattery

The CubeBattery structure has the following fields:

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

Table 143: CubeBattery JSON structure

49.2.2 CubeConnectionLost

“Indicates that the connection subscribed through ConnectCube has been lost.”

See also *ObjectConnectionState*

The ConnectCubeRequest has no fields.

49.3. CONNECT CUBE

“Attempt to connect to a cube. If a cube is currently connected, this will do nothing.”

Post: “/v1/disconnect_cube”

49.3.1 Request

The ConnectCubeRequest has no fields.

49.3.2 Response

The ConnectCubeResponse type has the following fields:

Field	Type	Description
<i>factory_id</i>	string	The identifier for the cube. This is built into the cube.
<i>object_id</i>	uint32	The identifier of the cube that we connected with. This is Vector’s internal identifier, and only the preferred cube is assigned one.
<i>status</i>	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.
<i>success</i>	bool	True if Vector was able to successfully connect, via Bluetooth LE, with the cube.

Table 144: ConnectCubeResponse JSON structure

49.4. CUBES AVAILABLE

Have Vector scan for cubes via Bluetooth LE and report the ones heard.

Post: “/v1/cubes_available”

49.4.1 Request

The CubesAvailableRequest has no fields.

49.4.2 Response

The CubesAvailableResponse is sent to indicate whether the action successfully completed or not.

This structure has the following fields:

Field	Type	Description
<i>factory_ids</i>	string[]	A list of the cubes that were seen via Bluetooth LE. The cubes internal identifier (it's factor id) is sent.
<i>status</i>	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 145:
CubesAvailableResponse JSON structure

49.5. DISCONNECT CUBE

“Requests a disconnection from the currently connected cube.”

Post: “/v1/disconnect_cube”

49.5.1 Request

The DisconnectCubeRequest has no fields.

49.5.2 Response

The DisconnectCubeResponse is sent to indicate whether the action successfully completed or not.

This structure has the following fields:

Field	Type	Description
<i>status</i>	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 146:
DisconnectCubeResponse JSON structure

49.6. DOCK WITH CUBE

“Tells Vector to dock with a light cube with [an optional] given approach angle and distance.”

“While docking with the cube, Vector will use path planning.”

This action requires the use of the wheels (tracks). “Actions that use the wheels cannot be performed at the same time, otherwise you may see a TRACKS_LOCKED error.”

Post: “/v1/dock_with_cube”

49.6.1 Request

The DockWithCubeRequest structure has the following fields:

Field	Type	Units	Description	Table 147: DockWithCubeRequest JSON structure
<i>alignment_type</i>	AlignmentType		“Which part of the robot to align with the object.”	
<i>approach_angle_rad</i>	float	radians	“The angle to approach the cube from. For example, 180 degrees will cause Vector to drive past the cube and approach it from behind.”	
<i>distance_from_marker_mm</i>	float	mm	“The distance from the object to stop. This is the distance between the origins.” 0mm to dock.	
<i>id_tag</i>	int32		This is an action tag that can be assigned to this request and used later to cancel the action. <i>Optional</i> .	
<i>motion_prof</i>	PathMotionProfile		Modifies how Vector should approach the cube. <i>Optional</i> .	
<i>num_retries</i>	int32		Maximum of times to attempt to reach the object. A retry is attempted if Vector is unable to reach the target object.	
<i>object_id</i>	int32		The identifier of the object to dock with.	
<i>use_approach_angle</i>	bool		If true, Vector will approach the cube from the given approach angle; otherwise Vector will approach from the most convenient angle.	
<i>use_pre_dock_pose</i>	bool		“Whether or not to try to immediately [dock with the] object or first position the robot next to the object.” Recommended to set this to the same as <i>use_approach_angle</i> .	

49.6.2 Response

The DockWithCubeResponse is sent to indicate whether the action successfully completed or not.

This structure has the following fields:

Field	Type	Description	Table 148: DockWithCubeResponse JSON structure
<i>result</i>	ActionResult	An error code indicating the success of the action, the detailed reason why it failed, or that the action is still being carried out.	
<i>status</i>	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.	

49.7. FLASH CUBE LIGHTS

“Plays the default cube connection animation on the currently connected cube's lights.”

Note: “This [command] is intended for app level user surfacing of cube connectivity, not for SDK cube light control.”

Post: “/v1/flash_cube_lights”

49.7.1 Request

The FlashCubeLightsRequest has no fields.

49.7.2 Response

The FlashCubeLightsResponse is sent to indicate whether the action successfully completed or not.

This structure has the following fields:

Field	Type	Description
<i>status</i>	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 149:
FlashCubeLightsResponse JSON structure

49.8. FORGET PREFERRED CUBE

“Forget the robot's preferred cube. This will cause the robot to connect to the cube with the highest RSSI (signal strength) next time a connection is requested. Saves this preference to disk. The next cube that the robot connects to will become its preferred cube.”

See also Set Preferred Cube

Post: “/v1/forget_preferred_cube”

49.8.1 Request

The ForgetPreferredCubeRequest has no fields.

49.8.2 Response

The ForgetPreferredCubeResponse is sent to indicate whether the action successfully completed or not. This structure has the following fields:

Field	Type	Description
<i>status</i>	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 150:
ForgetPreferredCubeResponse JSON structure

49.9. PICKUP OBJECT

“Instruct the robot to pick up the supplied object.” “While picking up the cube, Vector will use path planning.”

“Note that actions that use the wheels cannot be performed at the same time, otherwise you may see a TRACKS_LOCKED error.”

49.9.1 Request

The PickupObjectRequest structure has the following fields:

Field	Type	Units	Description	Table 151: PickupObjectRequest JSON structure
<i>approach_angle_rad</i>	float	radians	“The angle to approach the cube from. For example, 180 degrees will cause Vector to drive past the cube and approach it from behind.”	
<i>id_tag</i>	int32		This is an action tag that can be assigned to this request and used later to cancel the action. <i>Optional.</i>	
<i>motion_prof</i>	PathMotionProfile		<i>zOptional.</i>	
<i>num_retries</i>	int32		Maximum of times to attempt to reach the object. A retry is attempted if Vector is unable to reach the target object.	
<i>object_id</i>	int32		The identifier of the object to pick up. ‘Negative value means currently selected object’	
<i>use_approach_angle</i>	bool		If true, Vector will approach the cube from the given approach angle; otherwise Vector will approach from the most convenient angle.	
<i>use_pre_dock_pose</i>	bool		“Whether or not to try to immediately pick up an object or first position the robot next to the object.”	

49.9.2 Response

The PickupObjectResponse is sent to indicate whether the action successfully completed or not. This structure has the following fields:

Field	Type	Description	Table 152: PickupObjectResponse JSON structure
<i>result</i>	ActionResult	An error code indicating the success of the action, the detailed reason why it failed, or that the action is still being carried out.	
<i>status</i>	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.	

49.10. PLACE OBJECT ON GROUND HERE

“Ask Vector to place the object he is carrying on the ground at the current location.”

49.10.1 Request

The PlaceObjectOnGroundRequest structure has the following fields:

Field	Type	Units	Description
<i>id_tag</i>	int32		This is an action tag that can be assigned to this request and used later to cancel the action. <i>Optional.</i>
<i>num_retries</i>	int32		Maximum of times to attempt to reach the object. A retry is attempted if Vector is unable to reach the target object.

Table 153:
PlaceObjectOnGroundRequest JSON structure

49.10.2 Response

The PlaceObjectOnGroundResponse is sent to indicate whether the action successfully completed or not. This structure has the following fields:

Field	Type	Description
<i>result</i>	ActionResult	An error code indicating the success of the action, the detailed reason why it failed, or that the action is still being carried out.
<i>status</i>	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 154:
PlaceObjectOnGroundResponse JSON structure

49.11. POP A WHEELIE

“Tell Vector to ‘pop a wheelie’ using his cube.” Vector will approach the cube, then “push down on [it] with [his] lift, to start the wheelie.”

49.11.1 Request

The PopAWheelieRequest structure has the following fields:

Field	Type	Units	Description
<i>approach_angle_rad</i>	float	radians	“The angle to approach the cube from. For example, 180 degrees will cause Vector to drive past the cube and approach it from behind.”
<i>id_tag</i>	int32		This is an action tag that can be assigned to this request and used later to cancel the action. <i>Optional.</i>
<i>motion_prof</i>	PathMotionProfile		<i>zOptional.</i>
<i>num_retries</i>	int32		Maximum of times to attempt to reach the object. A retry is attempted if Vector is unable to reach the target object.
<i>object_id</i>	int32		The identifier of the object to used to pop a wheelie. Negative value means currently selected object’
<i>use_approach_angle</i>	bool		If true, Vector will approach the cube from the given approach angle; otherwise Vector will approach from the most convenient angle.
<i>use_pre_dock_pose</i>	bool		“Whether or not to try to immediately [use the] object or first position the robot next to the object.” Recommended to set this to the same as <i>use_approach_angle</i> .

Table 155:
PopAWheelieRequest
JSON structure

49.11.2 Response

The PopAWheelieResponse is sent to indicate whether the action successfully completed or not.

This structure has the following fields:

Field	Type	Description
<i>result</i>	ActionResult	An error code indicating the success of the action, the detailed reason why it failed, or that the action is still being carried out.
<i>status</i>	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 156:
PopAWheelieResponse
JSON structure

49.12. ROLL BLOCK

“Make Vector roll his block, regardless of relative position and orientation.” This triggers a behaviour, where Vector will look for his block, then “move into position as necessary based on relative distance and orientation.”

See also roll object.

Post: “/v1/roll_block”

49.12.1 Request

The RollBlockRequest has no fields.

49.12.2 Response

The RollBlockResponse structure has the following fields:

Table 157:
RollBlockResponse
JSON structure

Field	Type	Description
<i>result</i>	BehaviorResults	
<i>status</i>	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.

49.13. ROLL OBJECT

“Tell Vector to roll his cube.” This triggers an action.

49.13.1 Request

The RollObjectRequest structure has the following fields:

Field	Type	Units	Description	Table 158: RollObjectRequest JSON structure
<i>approach_angle_rad</i>	float	radians	“The angle to approach the cube from. For example, 180 degrees will cause Vector to drive past the cube and approach it from behind.”	
<i>id_tag</i>	int32		This is an action tag that can be assigned to this request and used later to cancel the action. <i>Optional</i> .	
<i>motion_prof</i>	PathMotionProfile		<i>zOptional</i> .	
<i>num_retries</i>	int32		Maximum of times to attempt to reach the object. A retry is attempted if Vector is unable to reach the target object.	
<i>object_id</i>	int32		The identifier of the object to roll. ‘Negative value means currently selected object’	
<i>use_approach_angle</i>	bool		If true, Vector will approach the cube from the given approach angle; otherwise Vector will approach from the most convenient angle.	
<i>use_pre_dock_pose</i>	bool		“Whether or not to try to immediately [roll the] object or first position the robot next to the object.” Recommended to set this to the same as <i>use_approach_angle</i> .	

49.13.2 Response

The RollObjectResponse is sent to indicate whether the action successfully completed or not. This structure has the following fields:

Field	Type	Description	Table 159: RollObjectResponse JSON structure
<i>result</i>	ActionResult	An error code indicating the success of the action, the detailed reason why it failed, or that the action is still being carried out.	
<i>status</i>	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.	

49.14. SET CUBE LIGHTS

“Set each of the lights on the currently connected cube based on two RGB values each and timing data for how to transition between them.”

“Sets each LED on [Vector]’s cube. Two states are specified designated ‘on’ and ‘off’, each with a color, duration, and state transition time.”

See also the Chapter 18, *Cube Animations*.

49.14.1 Request

The SetCubeLightsRequest event is used TBD. The structure has the following fields:

Table 160:
SetCubeLightsRequest
JSON structure

Field	Type	Units	Description
<i>object_id</i>	uint32		The internal id for the cube.
<i>make_relative</i>	MakeRelativeMode		<i>Should be off(1)</i>
<i>off_color</i>	array of uint32[]		Each color corresponds to each of the 4 cube lights. Each color is represented as four values (red, green, blue, and alpha), in the range of 0..255.
<i>off_period_ms</i>	uint32[]	ms	The “off” duration for each of the 4 cube lights. This is the duration to show each cube light in its corresponding “off” color (in <i>off_color</i>).
<i>offset</i>	int32[4]		<i>recommend: set t four 0’s.</i>
<i>on_color</i>	array of uint32[]		Each color corresponds to each of the 4 cube lights. Each color is represented as four values (red, green, blue, and alpha), in the range of 0..255.
<i>on_period_ms</i>	uint32[]	ms	The “on” duration for each of the 4 cube lights. This is the duration to show each cube light in its corresponding “on” color (in <i>onColors</i>).
<i>relative_to_x</i>	float		Should be 0.0
<i>relative_to_y</i>	float		Should be 0.0
<i>rotate</i>	boolean		? Possibly to have the colors be assigned to the next clockwise (or counterclockwise) light periodically? Should be <i>false</i>
<i>transition_off_period_ms</i>	uint32[]	ms	The time (in ms) to transition from the on color to the off color.
<i>transition_on_period_ms</i>	uint32[]	ms	The time (in ms) to transition from the off color to the on color

The `MakeRelativeMode` is used to indicate how Vector should align with the object. The enumeration has the following named values:

Name	Value	Description	Table 161: <i>MakeRelativeMode Enumeration</i>
<code>UNKNOWN</code>	0		
<code>OFF</code>	1		
<code>BY_CORNER</code>	2		
<code>BY_SIDE</code>	3		

49.14.2 Response

The `SetCubeLightsResponse` is sent to indicate whether the action successfully completed or not.

This structure has the following fields:

Field	Type	Description	Table 162: <i>SetCubeLightsResponse JSON structure</i>
<code>status</code>	<code>ResponseStatus</code>	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.	

49.15. SET PREFERRED CUBE

“Set the robot’s preferred cube and save it to disk. The robot will always attempt to connect to this cube if it is available. This is only used in simulation ~~for now~~.”

Post: “/v1/set_preferred_cube”

49.15.1 Request

The `SetPreferredCubeRequest` structure has the following fields:

Field	Type	Units	Description	Table 163: <i>SetPreferredCubeRequest JSON structure</i>
<code>factory_id</code>	string		The identifier of the cube to use. This is built into the cube.	

49.15.2 Response

The `SetPreferredCubeResponse` is sent to indicate whether the action successfully completed or not.

This structure has the following fields:

Field	Type	Description	Table 164: <i>SetPreferredCubeResponse JSON structure</i>
<code>status</code>	<code>ResponseStatus</code>	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.	

50. DISPLAY

This section describes commands that are used to display imagery on Vector's LCD.

50.1. EVENTS

50.1.1 MirrorModeDisabled

The “MirrorModeDisabled” event is sent (see TBD) “if MirrorMode (camera feed displayed on face) is currently enabled but is automatically being disabled.”

The `MirrorModeDisabled` structure has no fields.

50.2. DISPLAY IMAGE RGB

“Sets screen (Vector's face) to” display the passed image.

Post: “/v1/display_face_image_rgb”

50.2.1 Request

The `DisplayFaceImageRGBRequest` structure has the following fields:

Field	Type	Units	Description
<code>duration_ms</code>	<code>uint32</code>	<code>ms</code>	“How long to display the image on the face.”
<code>face_data</code>	<code>bytes</code>		The raw data for the image to display. The LCD is 184x56, with RGB565 pixels (16 bits/pixel).
<code>interrupt_running</code>	<code>bool</code>		“If this image should overwrite any current images on the face.”

Table 165:
`DisplayFaceImageRGBRequest` JSON structure

50.2.2 Response

The `DisplayFaceImageRGBResponse` structure has the following fields:

Field	Type	Description
<code>status</code>	<code>ResponseStatus</code>	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 166:
`DisplayFaceImageRGBResponse` JSON structure

50.3. ENABLE MIRROR MODE

“When enabled, camera feed will appear on the robot’s face, along with any detections” (if enabled).

Post: “/v1/enable_mirror_mode”

50.3.1 Request

The EnableMirrorModeRequest message has the following fields:

Field	Type	Description
<i>enable</i>	bool	If true, enables displaying the camera feed (and detections) on the LCD.

Table 167:
EnableMirrorModeRequest JSON structure

50.3.2 Response

The EnableMirrorModeResponse structure has the following fields:

Field	Type	Description
<i>status</i>	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 168:
EnableMirrorModeResponse JSON structure

50.4. SET EYE COLOR

This is used to set Vector’s current eye color. See also preferences.

Post: “/v1/set_eye_color”

50.4.1 Request

The SetEyeColorRequest has the following fields:

Field	Type	Description
<i>hue</i>	float	The hue to set Vector’s eyes to.
<i>saturation</i>	float	The saturation of the color to set Vector’s eyes to.

Table 169:
SetEyeColorRequest JSON structure

50.4.2 Response

The SetEyeColorResponse structure has the following fields:

Field	Type	Description
<i>status</i>	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 170:
SetEyeColorResponse JSON structure

51. 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 16, 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.

To interact with faces, see section TBD.

see also onboarding

51.1. ENUMERATIONS

51.1.1 FaceEnrollmentResult

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

Table 171:
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.
UNKNOWN_FAILURE	9	

51.1.2 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
<i>EXPRESSION_UNKNOWN</i>	0	The facial expression could not be estimated. Note: this could be because the facial expression estimation is disabled.
<i>EXPRESSION_NEUTRAL</i>	1	The face does not appear to have any particular expression.
<i>EXPRESSION_HAPPINESS</i>	2	The face appears to be happy
<i>EXPRESSION_SURPRISE</i>	3	The face appears to be surprised.
<i>EXPRESSION_ANGER</i>	4	The face appears
<i>EXPRESSION_SADNESS</i>	5	The face appears to be sad.

Table 172:
FacialExpression
Enumeration

51.2. STRUCTURES

51.2.1 RobotRenamedEnrolledFace

The RobotRenamedEnrolledFace structure has the following fields:

Field	Type	Description
<i>face_id</i>	int32	The identifier code for the face.
<i>name</i>	string	The name now associated with the face.

Table 173:
RobotRenamedEnrolled
Face JSON structure

51.3. EVENTS

51.3.1 FaceEnrollmentComplete

The FaceEnrollmentComplete structure has the following fields:

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

Table 174:
FaceEnrollmentComplete
JSON structure

51.3.2 Meet Victor Face Scan Complete

The `MeetVictorFaceScanComplete` structure has no fields.

51.3.3 Meet Victor Face Scan Started

The `MeetVictorFaceScanStarted` structure has no fields.

51.3.4 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:

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

Table 175:
`RobotChangedObservedFaceID` JSON structure

51.3.5 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:

Field	Type	Description
<code>face_id</code>	<code>int32</code>	The identifier code for the face; negative if the face is not recognized, positive if it has been recognized.
<code>expression</code>	<code>FacialExpression</code>	The estimated facial expression seen on the face.
<code>expression_values</code>	<code>uint32[]</code>	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.
<code>img_rect</code>	<code>CladRect</code>	The area within the camera view holding the face.
<code>name</code>	<code>string</code>	The name associated with the face (if recognized). Empty if a name is not known.
<code>pose</code>	<code>PoseStruct</code>	The position and orientation of the face.
<code>left_eye</code>	<code>CladPoint[]</code>	A polygon outlining the left eye, with respect to the image rectangle.
<code>mouth</code>	<code>CladPoint[]</code>	A polygon outlining the mouth; the coordinates are in the camera image.
<code>nose</code>	<code>CladPoint[]</code>	A polygon outlining the nose; the coordinates are in the camera image.
<code>right_eye</code>	<code>CladPoint[]</code>	A polygon outlining the right eye; the coordinates are in the camera image.
<code>timestamp</code>	<code>uint32</code>	The time that the most recent facial information was obtained. The format is milliseconds since Vector's epoch.

Table 176:
`RobotObservedFace` JSON structure

51.4. CANCEL FACE ENROLLMENT

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

post: “/v1/cancel_face_enrollment”

51.4.1 Request

The CancelFaceEnrollmentRequest structure has no fields.

51.4.2 Response

The CancelFaceEnrollmentResponse has the following fields:

Field	Type	Description
<i>status</i>	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 177:
CancelFaceEnrollmentResponse JSON structure

51.5. 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”

51.5.1 Request

The EnableFaceDetectionRequest structure has the following fields:

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

Table 178:
EnableFaceDetectionRequest JSON structure

51.5.2 Response

The EnableFaceDetectionResponse has the following fields:

Field	Type	Description
<code>status</code>	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 179:
EnableFaceDetectionResponse JSON structure

51.6. ERASE ALL ENROLLED FACES

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

post: “/v1/erase_all_enrolled_faces”

51.6.1 Request

The EraseAllEnrolledFacesRequest structure has no fields.

51.6.2 Response

The EraseAllEnrolledFacesResponse has the following fields:

Field	Type	Description
<i>status</i>	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 180:
EraseAllEnrolledFacesResponse JSON structure

51.7. ERASE ENROLLED FACE BY ID

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

post: “/v1/erase_enrolled_face_by_id”

51.7.1 Request

The EraseEnrolledFaceByIDRequest structure has the following fields:

Field	Type	Description
<i>face_id</i>	int32	The identifier code for the face to erase.

Table 181:
EraseEnrolledFaceByIDRequest JSON structure

51.7.2 Response

The EraseEnrolledFaceByIDResponse has the following fields:

Field	Type	Description
<i>status</i>	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 182:
EraseEnrolledFaceByIDResponse JSON structure

51.8. FIND FACES

This causes Vector to look around for faces. He does this by turning in place and moving his head up and down. This is carried out by the TBD behaviour.

post: “/v1/find_faces”

51.8.1 Request

The FindFacesRequest structure has no fields.

51.8.2 Response

The FindFacesResponse structure has the following fields:

Table 183:
FindFacesResponse
JSON structure

Field	Type	Description
<i>result</i>	BehaviorResults	
<i>status</i>	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.

51.9. 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”

51.9.1 Request

The RequestEnrolledNamesRequest structure has no fields.

51.9.2 Response

The RequestEnrolledNamesRequest structure has the following fields:

Field	Type	Description
<i>faces</i>	LoadedKnownFace[]	An array of the faces that are associated with names.
<i>status</i>	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 184:
RequestEnrolledNames
Response JSON
structure

The LoadedKnownFace structure has the following fields:

Field	Type	Units	Description
<i>face_id</i>	int32		The identifier code for the face.
<i>last_seen_seconds_since_epoch</i>	int64	seconds	The timestamp of the time the face was last seen. The format is unix time: seconds since 1970, in UTC?
<i>name</i>	name		The name associated with the face.
<i>seconds_since_first_enrolled</i>	int64	seconds	The number of seconds since the face was first associated with a name and entered into the known faces database.
<i>seconds_since_last_seen</i>	int64	seconds	The number of seconds since the face was last seen
<i>seconds_since_last_updated</i>	int64	seconds	The number of seconds since (?) the name associated with the face was last changed.(?)

Table 185:
LoadedKnownFace
JSON structure

51.10. 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”

51.10.1 Request

The SetFaceToEnrollRequest structure has the following fields:

Field	Type	Description	Table 186: SetFaceToEnrollRequest JSON structure
<i>name</i>	string	The name to associate with the face.	
<i>observed_id</i>	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.	
<i>save_id</i>	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 <i>observedID</i> for the ID.	
<i>save_to_robot</i>	bool	If true, “save to robot’s NVStorage when done (NOTE: will (re)save everyone enrolled!)”	
<i>say_name</i>	bool	If true, “play say-name/celebration animations on success before completing.”	
<i>use_music</i>	bool	If true, “starts special music during say-name animations (will leave music playing!)”	

51.10.2 Response

The SetFaceToEnrollResponse has the following fields:

Field	Type	Description	Table 187: SetFaceToEnrollResponse JSON structure
<i>status</i>	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.	

51.11. UPDATE ENROLLED FACE BY ID

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

post: “/v1/update_enrolled_face_by_id”

51.11.1 Request

The UpdateEnrolledFaceByIDRequest structure has the following fields:

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

Table 188:
UpdateEnrolledFaceByIDRequest JSON structure

51.11.2 Response

The UpdateEnrolledFaceByIDResponse has the following fields:

Field	Type	Description
<i>status</i>	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 189:
UpdateEnrolledFaceByIDResponse JSON structure

52. 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 25 *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.

52.1. ENUMERATIONS

52.1.1 UserEntitlement

The UserEntitlement enumeration has the following named values:

Name	Value	Description
KICKSTARTER_EYES	0	<i>Note: This was an entitlement that was explored, but not used.</i>

Table 190:
UserEntitlement
Enumeration

52.2. GET FEATURE FLAG

The request the current setting of a feature flag.

post: “/v1/feature_flag”

52.2.1 Request

The FeatureFlagRequest message has the following fields:

Field	Type	Description
feature_name	string	

Table 191:
FeatureFlagRequest
JSON structure

52.2.2 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 192:
FeatureFlagResponse
JSON structure

52.3. GET FEATURE FLAG LIST

To get a list of the current feature flags.

post: “/v1/feature_flag_list”

52.3.1 Request

The following is streamed... to the robot?

Field	Type	Description
<i>request_list</i>	string	

Table 193:
FeatureFlagListRequest
JSON structure

52.3.2 Response

The FeatureFlagListResponse type has the following fields:

Field	Type	Description
<i>list</i>	string[]	An array of the feature flags
<i>status</i>	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 194:
FeatureFlagListResponse
JSON structure

52.4. UPDATE USER ENTITLEMENTS

UpdateUserEntitlements

Post: “/v1/update_user_entitlements”

52.4.1 Request

The UpdateUserEntitlementsRequest has the following fields:

Field	Type	Description	Table 195: JSON Parameters for UpdateUserEntitlementsRequest
<i>user_entitlements</i>	UserEntitlementsConfig		

The UserEntitlementsConfig has the following fields:

Field	Type	Description	Table 196: JSON Parameters for UserEntitlementsConfig
<i>kickstarter_eyes</i>	bool		

52.4.2 Response

The UpdateUserEntitlementsResponse type has the following fields:

Field	Type	Description	Table 197: UpdateUserEntitlements Response JSON structure
<i>code</i>	resultCode		
<i>doc</i>	Jdoc		
<i>status</i>	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.	

53. IMAGE PROCESSING

See also Faces, for detecting and recognizing faces, and enabling the features

53.1. ENUMERATIONS

53.1.1 ImageEncoding

The ImageEncoding is used to describe the format of the image data contained in the chunk. The enumeration has the following named values:

Name	Value	Description	Table 198: ImageEncoding Enumeration
<i>NONE_IMAGE_ENCODING</i>	0	Image is not encoded. TBD: does this mean no image?	
<i>RAW_GRAY</i>	1	“No compression”	
<i>RAW_RGB</i>	2	“no compression, just [RGBRGBRG...]”	
<i>YUYV</i>	3		
<i>YUV420SP</i>	4		
<i>BAYER</i>	5		
<i>JPEG_GRAY</i>	6		
<i>JPEG_COLOR</i>	7		
<i>JPEG_COLOR_HALF_WIDTH</i>	8		
<i>JPEG_MINIMIZED_GRAY</i>	9	“Minimized grayscale JPEG - no header, no footer, no byte stuffing”	
<i>JPEG_MINIMIZED_COLOR</i>	10	“Minimized grayscale JPEG – no header, no footer, no byte stuffing, with added color data.”	

53.2. EVENTS

53.2.1 VisionModesAutoDisabled

This event is “sent when vision modes are automatically disabled due to the SDK no longer having control of the robot.”

The VisionModesAutoDisabled structure has no fields.

53.3. CAMERA FEED

“Request a camera feed from the robot.”

Post: “/v1/camera_feed”

53.3.1 Request

The CameraFeedRequest has no fields.

53.3.2 Response

The response is a stream of the following messages.

The CameraFeedResponse structure has the following fields:

Table 199:
CameraFeedResponse
JSON structure

Field	Type	Description
<i>data</i>	bytes	The bytes of the image
<i>frame_time_stamp</i>	uint32	
<i>image_encoding</i>	ImageEncoding	
<i>image_id</i>	uint32	
<i>status</i>	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.

53.4. CAPTURE SINGLE IMAGE

“Request a single image to be captured and sent from the robot.”

Post: “/v1/capture_single_image”

53.4.1 Request

The CaptureSingleImageRequest has no fields.

53.4.2 Response

The CaptureSingleImageResponse structure has the following fields:

Field	Type	Description
<i>data</i>	bytes	The bytes of the image
<i>frame_time_stamp</i>	uint32	
<i>image_encoding</i>	ImageEncoding	
<i>image_id</i>	uint32	
<i>status</i>	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.

53.5. ENABLE IMAGE STREAMING

“Toggle image streaming at the given resolution”

Post: “/v1/enable_image_streaming”

53.5.1 Request

The EnableImageStreamingRequest type has the following fields:

Field	Type	Description
<i>enable</i>	bool	True if a stream of images from the camera should be sent.

53.5.2 Response

The EnableImageStreamingResponse has the following fields:

Field	Type	Description
<i>status</i>	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.

53.6. ENABLE MARKER DETECTION

This enables and disables the processing of custom marker symbols and generating events when a marker symbol is seen. If enabled, when an marker symbol is seen, the RobotObservedObject event will be sent.

Note: The custom marker detection may remain internally enabled, even if disabled by the SDK, “if another subscriber (including one internal to the robot) requests this vision mode be active.”

Post: “/v1/enable_marker_detection”

53.6.1 Request

The EnableMarkerDetectionRequest has the following fields:

Field	Type	Description
enable	bool	If true, enable search for marker symbols, and generating events when they are detected,

Table 203: JSON Parameters for EnableMarkerDetection Request

53.6.2 Response

The EnableMarkerDetectionResponse 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 204: EnableMarkerDetection Response JSON structure

53.7. ENABLE MOTION DETECTION

Enables detection visual motion, and sending RobotObservedMotion (not implemented) events.

Post: “/v1/enable_motion_detection”

53.7.1 Request

The EnableMotionDetectionRequest structure has the following fields:

Field	Type	Description
enable	bool	True if RobotObservedMotion events should be sent.

Table 205:
EnableMotionDetectionRequest JSON structure

53.7.2 Response

The EnableMotionDetectionResponse 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 206:
EnableMotionDetectionResponse JSON structure

53.8. IS IMAGE STREAMING ENABLED

“Request whether or not image streaming is enabled on the robot”

Post: “/v1/is_image_streaming_enabled”

53.8.1 Request

The IsImageStreamingRequest has no fields.

53.8.2 Response

The IsImageStreamingResponse “indicates whether or not image streaming is enabled on the robot.”

The structure has the following fields:

Field	Type	Description
enable	bool	True if image streaming is enabled, false otherwise

Table 207:
IsImageStreamingResponse JSON structure

54. INTERACTIONS WITH OBJECTS

These commands are used to interact with faces and objects. These initiate behaviours.

- Some behaviours can be assigned a tag that can be used to cancel it later.
- Some behaviours accept a parameter to modify their motion profile.
- Behaviour results value

Actions

- Actions can be assigned a tag that can be used to cancel it later.
- Action results value

See also actions & behaviors, cube

54.1. STRUCTURES

54.1.1 PathMotionProfile

This structure contains “all the information relevant to how a path should be modified or traversed.”

Table 208:
PathMotionProfile
JSON structure

Field	Type	Units	Description
<i>accel_mmmps2</i>	float	<i>mm/sec²</i>	How fast Vector should accelerate to achieve the speed.
<i>decel_mmmps2</i>	float	<i>mm/sec²</i>	How fast Vector should decelerate to the speed.
<i>is_custom</i>	bool		
<i>dock_accel_mmmps2</i>	float	<i>mm/sec²</i>	How fast Vector should accelerate when performing the docking procedure.
<i>dock_decel_mmmps2</i>	float	<i>mm/sec²</i>	How fast Vector should decelerate when performing the docking procedure.
<i>dock_speed_mmmps</i>	float	<i>mm/sec</i>	The speed that Vector should perform the docking procedure at.
<i>point_turn_accel_mmmps2</i>	float	<i>mm/sec²</i>	How fast Vector should accelerate when turning (in place).
<i>point_turn_decel_mmmps2</i>	float	<i>mm/sec²</i>	How fast Vector should decelerate when turning (in place).
<i>point_turn_speed_mmmps</i>	float	<i>mm/sec</i>	The speed that Vector should perform a turn (in place) at.
<i>reverse_speed_mmmps</i>	float	<i>mm/sec</i>	How fast Vector should move when backing up
<i>speed_mmmps</i>	float	<i>mm/sec</i>	The speed that Vector should move along the path

54.2. DRIVE OFF CHARGER

This command directs Vector to drive off his charger – if he is on it. This will initiate a behavior.

Post: “/v1/drive_off_charger”

54.2.1 Request

The DriveOffChargerRequest structure has no fields.

54.2.2 Response

The DriveOffChargerResponse type has the following fields:

Field	Type	Description
<i>result</i>	BehaviorResults	
<i>status</i>	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 209:
DriveOffChargerResponse JSON structure

54.3. DRIVE ON CHARGER

This command directs Vector to drive onto his charger – if he is not already on it. “Vector will attempt to find the charger and, if successful, he will back onto it and start charging. Vector’s charger has a visual marker so that the robot can locate it for self-docking.” This will initiate a behaviour.

Post: “/v1/drive_on_charger”

54.3.1 Request

The DriveOnChargerRequest structure has no fields.

54.3.2 Response

The DriveOnChargerResponse type has the following fields:

Field	Type	Description
<i>result</i>	BehaviorResults	
<i>status</i>	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 210:
DriveOnChargerResponse JSON structure

54.4. GO TO OBJECT

“Tell Vector to drive to the specified object” (i.e. his cube). Note: custom objects “are not supported.” This initiates an action.

54.4.1 Request

The GoToObjectRequest structure has the following fields:

Field	Type	Units	Description	Table 211: GoToObjectRequest JSON structure
<i>id_tag</i>	int32		This is an action tag that can be assigned to this request and used later to cancel the action. <i>Optional.</i>	
<i>motion_prof</i>	PathMotionProfile		<i>Optional.</i>	
<i>num_retries</i>	int32		Maximum of times to attempt to reach the object. A retry is attempted if Vector is unable to reach the target object.	
<i>object_id</i>	int32		The identifier of the object to drive to. Note: custom objects “are not supported”	
<i>distance_from_object_origin_mm</i>	float	mm	“The distance from the object to stop. This is the distance between the origins. For instance, the distance from the robot’s origin (between Vector’s two front wheels) to the cube’s origin (at the center of the cube) is ~40mm.”	
<i>use_pre_dock_pose</i>	bool		Set this to false	

54.4.2 Response

The GoToObjectResponse is sent to indicate whether the action successfully completed or not. This structure has the following fields:

Field	Type	Description	Table 212: GoToObjectResponse JSON structure
<i>result</i>	ActionResult	An error code indicating the success of the action, the detailed reason why it failed, or that the action is still being carried out.	
<i>status</i>	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.	

54.5. TURN TOWARDS FACE

“Tell Vector to turn towards this face.” This initiates an action.

54.5.1 Request

The TurnTowardsFaceRequest structure has the following fields:

Field	Type	Description
<i>face_id</i>	int32	The identifier of the face to look for
<i>id_tag</i>	int32	This is an action tag that can be assigned to this request and used later to cancel the action. <i>Optional.</i>
<i>max_turn_angle_rad</i>	float	Recommend value of 180°
<i>num_retries</i>	int32	Maximum of times to attempt to reach the object. A retry is attempted if Vector is unable to reach the target object.

Table 213:
TurnTowardsFaceRequest JSON structure

54.5.2 Response

The TurnTowardsFaceResponse is sent to indicate whether the action successfully completed or not.

This structure has the following fields:

Field	Type	Description
<i>result</i>	ActionResult	An error code indicating the success of the action, the detailed reason why it failed, or that the action is still being carried out.
<i>status</i>	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 214:
TurnTowardsFaceResponse JSON structure

55. 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.

55.1. ENUMERATIONS

55.1.1 JdocType

The JdocType enumeration has the following named values:

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.

Table 215: JdocType Enumeration

Items of these types are described in more detail in Chapter 23.

55.2. STRUCTURES

55.2.1 JDoc

The Jdoc type has the following fields:

Field	Type	Description
client_meta	string	
doc_version	uint64	
fmt_version	uint64	
json_doc	string	

Table 216: Jdoc JSON structure

55.2.2 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 217: JSON NamedJdoc structure

55.3. EVENTS

55.3.1 JdocsChanged

The JdocsChanged message has the following fields:

Field	Type	Description	<i>Table 218: JSON JdocsChanged request structure</i>
<i>jdoc_types</i>	JdocType[]		

55.4. PULL JDOCS

Post: “/v1/pull_jdocs”

55.4.1 Request

The PullJdocsRequest has the following fields:

Field	Type	Description	<i>Table 219: JSON PullJdocsRequest structure</i>
<i>jdoc_types</i>	JdocType[]		

55.4.2 Response

The PullJdocResponse has the following fields:

Field	Type	Description	<i>Table 220: JSON PullJdocsResponse structure</i>
<i>named_jdocs</i>	NamedJdoc[]		
<i>status</i>	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.	

56. MISC, ACCESSORIES AND CUSTOM OBJECTS

56.1. UPLOAD DEBUG LOGS

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

Post: “/v1/upload_debug_logs”

56.1.1 Request

The UploadDebugLogsRequest structure has no fields.

56.1.2 Response

The UploadDebugLogsResponse structure has the following fields:

Field	Type	Description
<i>status</i>	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.
<i>url</i>	string	

Table 221:
UploadDebugLogsResponse JSON structure

57. MOTION CONTROL

This section describes commands to drive Vector, and to control his lift & head position. See also TBD interactions.

57.1. DRIVE STRAIGHT

This will initiate an action of Vector driving 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. [A] that use the wheels cannot be performed at the same time; otherwise you may see a TRACKS_LOCKED error.”

57.1.1 Request

The DriveStraightRequest has the following fields:

Field	Type	Units	Description	Table 222: DriveStraightRequest JSON structure
<i>dist_mm</i>	float	mm	The distance to drive. (Negative is backwards)	
<i>id_tag</i>	int32		This is an action tag that can be assigned to this request and used later to cancel the action. <i>Optional.</i>	
<i>is_absolute</i>	uint32		If 0, turn by <i>angle_rad</i> relative to the current orientation. If 1, turn to the absolute angle given by <i>angle_rad</i> .	
<i>num_retries</i>	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	
<i>should_play_animation</i>	bool		If true, “play idle animations whilst driving (tilt head, hum, animated eyes, etc.)”	
<i>speed_mmmps</i>	float	mm/sec	The speed to drive at. This should be positive.	

57.1.2 Response

The DriveStraightResponse has the following fields:

Field	Type	Description	Table 223: DriveStraightResponse JSON structure
<i>response</i>	ActionResult	Whether the action is able to be run. If not, an error code indicating why not.	
<i>status</i>	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.	

57.2. DRIVE WHEELS

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

57.2.1 Request

The DriveWheelsRequest has the following fields:

Field	Type	Units	Description	Table 224: DriveWheelsRequest JSON structure
<i>left_wheel_mmmps</i>	float	mm/sec	The initial speed to set the left wheel to.	
<i>left_wheel_mmmps2</i>	float	mm/sec ²	How fast to increase the speed of the left wheel.	
<i>right_wheel_mmmps</i>	float	mm/sec	The initial speed to set the right wheel to.	
<i>right_wheel_mmmps2</i>	float	mm/sec ²	How fast to increase the speed of the right wheel.	

To unlock the wheels, set all values to 0.

57.2.2 Response

The DriveWheelsResponse has the following fields:

Field	Type	Description	Table 225: DriveWheelsResponse JSON structure
<i>status</i>	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.	

57.3. GO TO POSE

“Tells Vector to drive to the specified pose and orientation.” This will initiate an action.

“In navigating to the requested pose, Vector will use path planning.

“Since the robot understands position by monitoring its tread movement, it does not understand movement in the z axis. This means that the only applicable elements of pose in this situation are position.x position.y and rotation.angle_z.

“Note that actions that use the wheels cannot be performed at the same time, otherwise you may see a TRACKS_LOCKED error.”

Post: “/v1/go_to_pose”

57.3.1 Request

The GoToPoseRequest structure has the following fields:

Field	Type	Units	Description
<i>id_tag</i>	int32		This is an action tag that can be assigned to this request and used later to cancel the action. <i>Optional.</i>
<i>motion_prof</i>	PathMotionProfile		
<i>num_retries</i>	int32		Maximum of times to attempt to reach the pose. A retry is attempted if Vector is unable to reach the target pose.
<i>rad</i>	float	radians	The angle to change orientation to.
<i>x_mm</i>	float	mm	The x-coordinate of the position to move to.
<i>y_mm</i>	float	mm	The y-coordinate of the position to move to.

Table 226:
GoToPoseRequest JSON structure

57.3.2 Response

The GoToPoseResponse is sent to indicate whether the action successfully completed or not. This structure has the following fields:

Field	Type	Description
<i>result</i>	ActionResult	An error code indicating the success of the action, the detailed reason why it failed, or that the action is still being carried out.
<i>status</i>	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 227:
GoToPoseResponse JSON structure

57.4. MOVE HEAD

Move Vector's head

57.4.1 Request

The `MoveHeadRequest` has the following fields:

Field	Type	Units	Description
<code>speed_rad_per_sec</code>	float	radian/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 228:
`MoveHeadRequest`
JSON structure

57.4.2 Response

The `MoveHeadResponse` has the following fields:

Field	Type	Description
<code>status</code>	<code>ResponseStatus</code>	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 229:
`MoveHeadResponse`
JSON structure

57.5. MOVE LIFT

Move Vector's lift

57.5.1 Request

The `MoveLiftRequest` has the following fields:

Field	Type	Units	Description
<code>speed_rad_per_sec</code>	float	radian/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 230:
`MoveLiftRequest` JSON structure

57.5.2 Response

The `MoveLiftResponse` has the following fields:

Field	Type	Description
<code>status</code>	<code>ResponseStatus</code>	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 231:
`MoveLiftResponse` JSON structure

57.6. SET HEAD ANGLE

This will initiate an action to move Vector's head to a given angle.

57.6.1 Request

The SetHeadAngleRequest has the following fields:

Field	Type	Units	Description	Table 232: SetHeadAngleRequest JSON structure
<i>accel_rad_per_sec2</i>	float	radian/sec ²	How fast to increase the speed the head is moving at. Recommended value: 10 radians/sec ²	
<i>angle_rad</i>	float	radians	The target angle to move Vector's head to. This should be in the range -22.0° to 45.0°.	
<i>duration_sec</i>	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.”	
<i>id_tag</i>	int32		This is an action tag that can be assigned to this request and used later to cancel the action. <i>Optional.</i>	
<i>max_speed_rad_per_sec</i>	float	radian/sec	The maximum speed to move the head. (This clamps the speed from further acceleration.) Recommended value: 10 radians/sec	
<i>num_retries</i>	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	

57.6.2 Response

The SetHeadAngleResponse has the following fields:

Field	Type	Description	Table 233: SetHeadAngleResponse JSON structure
<i>response</i>	ActionResult	Whether the action is able to be run. If not, an error code indicating why not.	
<i>status</i>	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.	

57.7. SET LIFT HEIGHT

This will initiate an action to move Vector's lift to a given height.

57.7.1 Request

The SetLiftRequest has the following fields:

Field	Type	Units	Description
<i>accel_rad_per_sec2</i>	float	radian/sec ²	How fast to increase the speed the lift is moving at/ Recommended value: 10 radians/sec ²
<i>duration_sec</i>	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.”
<i>height_mm</i>	float	mm	The target height to raise the lift to. Note: the python API employs a different range for this parameter
<i>id_tag</i>	int32		This is an action tag that can be assigned to this request and used later to cancel the action. <i>Optional.</i>
<i>max_speed_rad_per_sec</i>	float	radian/sec	The maximum speed to move the lift at. (This clamps the speed from further acceleration.) Recommended value: 10 radians/sec
<i>num_retries</i>	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 234:
SetLiftRequest JSON
structure

57.7.2 Response

The SetLiftResponse has the following fields:

Field	Type	Description
<i>response</i>	ActionResult	Whether the action is able to be run. If not, an error code indicating why not.
<i>status</i>	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 235:
SetLiftResponse JSON
structure

57.8. STOP ALL MOTORS

Stop all motor commands for the head, lift and wheels

57.8.1 Request

The StopAllMotorsRequest structure has no fields.

57.8.2 Response

The StopAllMotorsResponse has the following fields:

Field	Type	Description	Table 236: StopAllMotorsResponse JSON structure
<i>status</i>	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.	

57.9. TURN IN PLACE

This command initiates an action to turn Vector 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.”

57.9.1 Request

The TurnInPlaceRequest has the following fields:

Field	Type	Units	Description	Table 237: TurnInPlaceRequest JSON structure
<i>accel_rad_per_sec2</i>	float	radian/sec ²	How fast to increase the speed the body is moving at	
<i>angle_rad</i>	float	radians	If <i>isAbsolute</i> 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.	
<i>id_tag</i>	int32		This is an action tag that can be assigned to this request and used later to cancel the action. <i>Optional</i> .	
<i>is_absolute</i>	uint32		If 0, turn by <i>angle_rad</i> relative to the current orientation. If 1, turn to the absolute angle given by <i>angle_rad</i> .	
<i>num_retries</i>	int32	<i>count</i>	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	
<i>speed_rad_per_sec</i>	float	radian/sec	The speed to move around the arc.	
<i>tol_rad</i>	float	<i>count</i>	“The angular tolerance to consider the action complete (this is clamped to a minimum of 2 degrees internally).”	

57.9.2 Response

The TurnInPlaceResponse has the following fields:

Field	Type	Description	Table 238: TurnInPlaceResponse JSON structure
<i>response</i>	ActionResult	Whether the action is able to be run. If not, an error code indicating why not.	
<i>status</i>	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.	

58. MOTION SENSING AND ROBOT STATE

This section describes the structures and events that describe the sensed motions.

The values are given in a “coordinate space is relative to Vector, where Vector's origin is the point on the ground between Vector's two front wheels. The X axis is Vector's forward direction, the Y axis is to Vector's left, and the Z axis is up.”

58.1. STRUCTURES

58.1.1 AccelData

This structure 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 Vector's head. The AccelData has the following fields:

Field	Type	Units	Description
x	float	mm/s ²	The acceleration along the accelerometers X axis.
y	float	mm/s ²	The acceleration along the accelerometers Y axis.
z	float	mm/s ²	The acceleration along the accelerometers Z axis.

Table 239: AccelData JSON structure

When at rest, there will be a constant 9810 mm/s² downward acceleration from gravity. This most likely will be distributed across multiple axes.

58.1.2 GyroData

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

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.

Table 240: GyroData JSON structure

58.1.3 ProxData

This structure is used to report the “time of flight” proximity sensor readings, as part of the RobotState structure.

“The proximity sensor is located near the bottom of Vector between the two front wheels, facing forward. The reported distance describes how far in front of this sensor the robot feels an obstacle is. The sensor estimates based on time-of-flight information within a field of view which the engine resolves to a certain quality value.”

The distance measurement may not be valid; “four additional flags are supplied by the engine to indicate whether this proximity data is considered valid for the robot’s internal pathfinding.” It is recommended that an application track the most recent proximity data from the robot, and the most recent proximity data which did not have the lift blocking.

The ProxData has the following fields:

Field	Type	Units	Description
<i>distance_mm</i>	uint32	<i>mm</i>	The distance to the object
<i>found_object</i>	bool		True if “the sensor detected an object in the valid operating range.”
<i>is_lift_in_fov</i>	bool		True if “Vector’s lift (or an object in the lift) is blocking the time-of-flight sensor. While the lift will send clear proximity signals, it’s not useful for object detection.”
<i>signal_quality</i>	float		An estimate of the “reliability of the measurement.” “The proximity sensor detects obstacles within a given field of view; this value represents the likelihood of the reported distance being a solid surface.”
<i>unobstructed</i>	bool		True if “the sensor has confirmed it has not detected anything up to its max range.” (The opposite of <i>found_object</i>)

Table 241: ProxData
JSON structure

58.1.4 TouchData

This structure is used to report the touch sensor readings, as part of the RobotState structure. The TouchData has the following fields:

Field	Type	Units	Description
<i>is_being_touched</i>	bool		True if Vector is currently being touched.
<i>raw_touch_value</i>	uint32		“Raw input from the touch sensor.”

Table 242: TouchData
JSON structure

58.2. EVENTS

58.2.1 RobotState

The RobotState structure is periodically in an *Event* message. The structure has the following fields:

Table 243: RobotState JSON structure

Field	Type	Units	Description
<i>accel</i>	AccelData		The accelerometer readings
<i>carrying_object_id</i>	int32		-1 if no object is being carried. Otherwise the identifier of the cube (or other object) being carried.
<i>carrying_object_on_top_id</i>	int32		<i>Not supported</i>
<i>gyro</i>	GyroData		The gyroscope readings
<i>head_angle_rad</i>	float	radian	The angle of Vector's head (how much it is tilted up or down).
<i>head_tracking_object_id</i>	int32		The identifier “of the object the head is tracking to.” If no object is being tracked, this will be -1.
<i>last_image_time_stamp</i>	uint32		“The robot's timestamp for the last image seen.” The format is milliseconds since Vector's epoch.
<i>left_wheel_speed_mmmps</i>	float	mm/s	The speed of Vector's left wheel.
<i>lift_height_mm</i>	float	mm	“Height of Vector's lift from the ground.”
<i>localized_to_object_id</i>	int32		The identifier “of the object that the robot is localized to.” If no object, this will be -1.
<i>pose</i>	PoseStruct		“The current pose (position and orientation) of Vector.”
<i>pose_angle_rad</i>	float	radian	“Vector's pose angle (heading in X-Y plane).”
<i>pose_pitch_rad</i>	float	radian	“Vector's pose pitch (angle up/down).”
<i>right_wheel_speed_mmmps</i>	float	mm/s	The speed of Vector's right wheel.
<i>prox_data</i>	ProxData		The time-of-flight proximity sensor readings.
<i>status</i>	uint32		A bit map of active states of Vector; the bits are described in the RobotStatus enumeration.
<i>touch_data</i>	TouchData		The touch sensor readings.

Note: all quotes above are from the python SDK.

59. ONBOARDING

The section describes the command and events used to introduce Vector to his new human, his human to Vector's features.

59.1. ENUMERATIONS

59.1.1 OnboardingPhase

The OnboardingPhase enumeration has the following named values:

Name	Value	Description	Table 244: OnboardingPhase Enumeration
<i>InvalidPhase</i>	0		
<i>Default</i>	1		
<i>LookAtPhone</i>	2		
<i>WakeUp</i>	3		
<i>LookAtUser</i>	4		
<i>TeachWakeWord</i>	5		
<i>TeachComeHere</i>	6		
<i>TeachMeetVictor</i>	7		

59.1.2 OnboardingPhaseState

The OnboardingPhaseState enumeration has the following named values:

Name	Value	Description	Table 245: OnboardingPhaseState Enumeration
<i>PhaseInvalid</i>	0		
<i>PhasePending</i>	1		
<i>PhaseInProgress</i>	2		
<i>PhaseComplete</i>	3		

59.2. EVENTS

59.2.1 Onboarding

The Onboarding event is sent as different stages of the onboarding process have been completed.
This structure has the following fields:

Field	Type	Description
<i>onboarding_1p0_charging_info</i>	Onboarding1p0ChargingInfo	
<i>onboarding_state</i>	OnboardingState	
<i>onboarding_wakeup_finished</i>	{}	This structure contains no fields.

Table 246: Onboarding JSON structure

59.2.2 Onboarding1p0ChargingInfo

This structure is used to report whether Vector needs to change, and an estimated duration. It is part of the Onboarding event structure. This structure has the following fields:

Field	Type	Units	Description
<i>needs_to_charge</i>	bool		If true, Vector needs to charge before onboarding can continue.
<i>on_charger</i>	bool		If true, Vector is on his charger, and there is power supplied to the charger.
<i>suggested_charger_time</i>	float		The estimated amount of time to charge Vector before completing the onboarding process.

Table 247:
Onboarding1p0ChargingInfoJSON structure

59.2.3 OnboardingState

The OnboardingState type has the following fields:

Field	Type	Description
<i>stage</i>	OnboardingStages	Where in the onboarding process we are

Table 248:
OnboardingState JSON structure

The OnboardingStages enumeration has the following named values:

Name	Value	Description
<i>NotStarted</i>	0	The onboarding process has not started yet.
<i>TimedOut</i>	1	The onboarding process has halted because an operation timed out.
<i>Complete</i>	3	The onboarding has completed.
<i>DevDoNothing</i>	4	

Table 249:
OnboardingStages Enumeration

59.3. ONBOARDING INPUT

Post: “/v1/send_onboarding_input”

59.3.1 Request

The OnboardingInputRequest has one (and only one) of the following fields:

Field	Type	Description	Table 250: OnboardingInputRequest JSON structure
<code>onboarding_charge_info_request</code>	{}	This is a request for charging information; it contains no fields.	
<code>onboarding_complete_request</code>	{}	This is a request to complete the onboarding; it contains no fields.	
<code>onboarding_mark_complete_and_exit</code>	{}	This contains no fields.	
<code>onboarding_restart</code>	{}	This is a request to restart the onboarding process; it contains no fields.	
<code>onboarding_skip_onboarding</code>	{}	This is a request to skip the onboarding; it contains no fields.	
<code>onboarding_phase_progress_request</code>	{}	This contains no fields.	
<code>onboarding_set_phase_request</code>	OnboardingSetPhaseRequest	See below.	
<code>onboarding_wake_up_request</code>	{}	This is a request to wake up Vector; it contains no fields.	
<code>onboarding_wake_up_started_request</code>	{}	This contains no fields.	

The OnboardingSetPhaseRequest type has the following fields:

Field	Type	Description	Table 251: OnboardingSetPhaseRequest JSON structure
<code>phase</code>	OnboardingPhase	The desired phase to be in	

59.3.2 Response

The OnboardingInputResponse has a status field one (and only one) of the remaining following fields (which will correspond to the one sent in the request):

Field	Type	Description	Table 252: OnboardingInputResponse JSON structure
<code>onboarding_charge_info_response</code>	OnboardingChargingInfoResponse	<i>See below</i>	
<code>onboarding_complete_response</code>	OnboardingCompleteResponse	<i>See below</i>	
<code>onboarding_phase_progress_response</code>	{}	<i>See below</i>	
<code>onboarding_set_phase_response</code>	OnboardingSetPhaseResponse	<i>See below.</i>	
<code>onboarding_wake_up_response</code>	OnboardingWakeUpResponse	<i>See below</i>	
<code>onboarding_wake_up_started_response</code>	OnboardingWakeupStartedResponse		
<code>status</code>	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.	

59.3.2.1 OnboardingChargingInfoResponse

This structure is used to report whether Vector needs to charge, and an estimated duration. It is part of the OnboardingInputResponse event structure. This structure has the following fields:

Field	Type	Units	Description	Table 253: OnboardingInputResponse structure
<code>needs_to_charge</code>	bool		If true, Vector needs to charge before onboarding can continue.	
<code>on_charger</code>	bool		If true, Vector is on his charger, and there is power supplied to the charger.	
<code>required_charge_time</code>	float		The estimated amount of time to charge Vector before completing the onboarding process.	
<code>status</code>	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: this structure is similar to the Onboarding1p0ChargingInfo structure. That structures is older, but retained as software had already been developed against it.

59.3.2.2 OnboardingCompleteResponse

The OnboardingCompleteResponse type has the following fields:

Field	Type	Description	Table 254: OnboardingCompleteResponse JSON structure
<code>completed</code>	bool	True if the onboarding has completed	

59.3.2.3 OnboardingPhaseProgressResponse

This structure is used to report how far we are in a particular phase of onboarding. It is part of the OnboardingInputResponse event structure. This structure has the following fields:

Field	Type	Units	Description
<i>last_set_phase</i>	OnboardingPhase		
<i>last_set_phase_state</i>	OnboardingPhaseState		
<i>percent_completed</i>	int32	%	How far we are in the phase.
<i>status</i>	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 255:
OnboardingPhaseProgres
sResponse structure

59.3.2.4 OnboardingSetPhaseResponse

The OnboardingSetPhaseResponse type has the following fields:

Field	Type	Units	Description
<i>last_set_phase</i>	OnboardingPhase		
<i>last_set_phase_state</i>	OnboardingPhaseState		
<i>status</i>	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 256:
OnboardingSetPhaseRes
ponse JSON structure

59.3.2.5 OnboardingWakeupResponse

The OnboardingWakeupResponse type has the following fields:

Field	Type	Description
<i>charging_info</i>	Onboarding1p0ChargingInfo	Whether or not Vector needs to charge after waking up.
<i>waking_up</i>	bool	True if Vector is waking up.

Table 257:
OnboardingWakeupRes
ponse JSON structure

59.3.2.6 OnboardingWakeupStartedResponse

The OnboardingWakeupStartedResponse type has the following fields:

Field	Type	Description
<i>already_started</i>	bool	True if the onboarding has completed

Table 258:
OnboardingWakeupStart
edResponse JSON
structure

59.4. ONBOARDING STATE

Post: “/v1/get_onboarding_state”

59.4.1 Request

The OnboardingStateRequest structure has no fields.

59.4.2 Response

The OnboardingStateResponse type has the following fields:

Field	Type	Description	
<i>onboarding_state</i>	OnboardingState	Where in the onboarding process we are.	Table 259: <i>OnboardingStateResponse</i> JSON structure
<i>status</i>	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.	

59.5. ONBOARDING COMPLETE REQUEST

59.5.1 Request

The OnboardingCompleteRequest structure has no fields.

59.5.2 Response

The OnboardingCompleteResponse type has the following fields:

Field	Type	Description	
<i>completed</i>	bool	True if the onboarding process has completed.	Table 260: <i>OnboardingCompleteResponse</i> JSON structure

59.6. ONBOARDING WAKE UP REQUEST

59.6.1 Request

The OnboardingWakeUpRequest structure has no fields.

59.6.2 Response

The OnboardingWakeUpResponse type has the following fields:

Field	Type	Description	
<i>already_started</i>	bool	True if the process of waking Vector up for onboarding has already been started.	Table 261: <i>OnboardingWakeUpResponse</i> JSON structure

59.7. ONBOARDING WAKE UP STARTED REQUEST

59.7.1 Request

The OnboardingWakeUpStartedRequest structure has no fields.

59.7.2 Response

The OnboardingWakeUpStartedResponse type has the following fields:

Field	Type	Description	Table 262: OnboardingWakeUpStar tedResponse JSON structure
<i>charging_info</i>	Onboarding1p0ChargingInfo	The state of Vectors initial charging	
<i>waking_up</i>	bool	True if TBD	

60. PHOTOS

This section describes the commands and queries related to accessing and managing photographs (and their thumbnail images) on Vector. For a description of the photos, see Chapter 16 *Image Processing*.

60.1. STRUCTURES

60.1.1 Photo Path

The PhotoPathMessage type has the following fields:

Field	Type	Description	Table 263: PhotoPathMessage JSON structure
<i>full_path</i>	string		
<i>success</i>	bool	True if the photograph exists; otherwise, the photograph does not exist.	

60.1.2 Thumbnail Path

The ThumbnailPathMessage type has the following fields:

Field	Type	Description	Table 264: ThumbnailPathMessage JSON structure
<i>full_path</i>	string		
<i>success</i>	bool	True if the thumbnail image exists; otherwise, the thumbnail does not exist.	

60.2. EVENTS

60.2.1 PhotoTaken

The PhotoTaken event is sent after Vector has taken a photograph and stored it. This structure has the following fields:

Field	Type	Description	Table 265: PhotoTaken JSON structure
<i>photo_id</i>	uint32	The identifier of the photograph taken.	

60.3. DELETE PHOTO

This command is used to delete a photograph and its thumbnail

Post: “/v1/delete_photo”

60.3.1 Request

The DeletePhotoRequest has the following fields:

Field	Type	Description
photo_id	uint32	The identifier of the photograph to delete.

Table 266:
DeletePhotoRequest
JSON structure

60.3.2 Response

The DeletePhotoResponse type 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.
success	bool	True if the photograph was successfully removed; otherwise there was an error.

Table 267:
DeletePhotoResponse
JSON structure

60.4. PHOTO

This command is used to retrieve the photograph’s image.

Post: “/v1/photo”

60.4.1 Request

The PhotoRequest structure has the following fields:

Field	Type	Description
photo_id	uint32	The identifier of the photograph to request.

Table 268:
PhotoRequest JSON
structure

60.4.2 Response

The PhotoResponse type has the following fields:

Field	Type	Description
image	bytes	The data that make up the photograph’s image
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.
success	bool	True if the photograph was successfully retrieved; otherwise there was an error.

Table 269:
PhotoResponse JSON
structure

60.5. PHOTOS INFO

This command is used to get the list of photographs available on Vector.

Post: “/v1/photos_info”

60.5.1 Request

The PhotosInfoRequest structure has no fields.

60.5.2 Response

The PhotosInfoResponse type has the following fields:

Field	Type	Description
<i>status</i>	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.
<i>photo_infos</i>	PhotoInfo[]	An array of information about the photographs available on Vector.

Table 270:
PhotosInfoResponse
JSON structure

The PhotoInfo type has the following fields:

Field	Type	Description
<i>photo_copied_to_app</i>	bool	True if the photograph has been downloaded to the application.
<i>photo_id</i>	uint32	The identifier of this photograph. This can be used to retrieve the thumbnail, photograph or to delete it.
<i>thumb_copied_to_app</i>	bool	True if the thumbnail image has been downloaded to the application.
<i>timestamp_utc</i>	uint32	The time that the photograph was taken. The format is unix time: seconds since 1970, in UTC.

Table 271: PhotoInfo
JSON structure

60.6. THUMBNAIL

This command is used to retrieve the thumbnail image of a photograph.

Post: “/v1/thumbnail”

60.6.1 Request

The `ThumbnailRequest` structure has the following fields:

Field	Type	Description
<code>photo_id</code>	<code>uint32</code>	The identifier of the photograph to request a thumbnail for.

Table 272:
`ThumbnailRequest`
JSON structure

60.6.2 Response

The `ThumbnailResponse` type has the following fields:

Field	Type	Description
<code>image</code>	<code>bytes</code>	The data that make up the thumbnail’s image
<code>status</code>	<code>ResponseStatus</code>	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.
<code>success</code>	<code>bool</code>	True if the thumbnail was successfully retrieved; otherwise there was an error.

Table 273:
`ThumbnailResponse`
JSON structure

61. 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 23 *Settings, Preferences, Features, and Statistics*. That chapter includes definitions for the following types:

- RobotSettingsConfig

61.1. STRUCTURES

61.1.1 AccountSettingsConfig

The AccountSettingsConfig type has the following fields:

Field	Type	Description	Table 274: AccountSetting JSON structure
app_locale	string		
data_collection	boolean		

61.2. UPDATE SETTINGS

Post: “/v1/update_settings”

61.2.1 Request

The UpdateSettingsRequest has the following fields:

Field	Type	Description	Table 275: UpdateSettingsRequest JSON structure
settings	RobotSettingsConfig		

61.2.2 Response

The UpdateSettingsResponse type has the following fields:

Field	Type	Description	Table 276: UpdateSettingsRespons e JSON structure
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.	

61.3. UPDATE ACCOUNT SETTINGS

Post: “/v1/update_account_settings”

61.3.1 Request

The UpdateAccountsSettingsRequest has the following fields:

Field	Type	Description
<i>account_settings</i>	AccountSettingsConfig	

Table 277: JSON Parameters for *UpdateAccountSettings Request*

61.3.2 Response

The UpdateAccountsSettingsResponse type has the following fields:

Field	Type	Description
<i>code</i>	resultCode	
<i>doc</i>	JDoc	
<i>status</i>	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 278: *UpdateAccountSettings Response* JSON structure

62. SOFTWARE UPDATES

These commands are siblings to the OTA Update and related commands in Chapter 11 Bluetooth LE protocol. However, they differ: in some cases, less information, in others present the same information in different ways.

62.1. ENUMERATIONS

62.1.1 UpdateStatus

The UpdateStatus enumeration has the following named values:

Table 279:
*UpdateStatus
Enumeration*

Name	Value	Description
<i>IN_PROGRESS_DOWNLOAD</i>	2	The software update is currently being downloaded.
<i>NO_UPDATE</i>	0	There are no software updates available.
<i>READY_TO_INSTALL</i>	1	The software update has been downloaded and is ready to install.

62.2. 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

62.3. CHECK UPDATE STATUS

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

Post: “/v1/check_update_status”

62.3.1 Request

The CheckUpdateStatusRequest structure has no fields.

62.3.2 Response

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

Field	Type	Description
<i>expected</i>	int64	The number of bytes expected to be downloaded
<i>progress</i>	int64	The number of bytes downloaded
<i>status</i>	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.
<i>update_status</i>	UpdateStatus	
<i>update_version</i>	string	

Table 280:
CheckUpdateStatusResponse JSON structure

62.4. UPDATE AND RESTART

Post: “/v1/update_and_restart”

62.4.1 Request

The UpdateAndRestartRequest structure has no fields.

62.4.2 Response

The UpdateAndRestartResponse has the following fields:

Field	Type	Description
<i>status</i>	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 281:
UpdateAndRestartResponse JSON structure

CHAPTER 15

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

63. 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.)

Element	Description & Notes
<i>appkey</i>	A base64 token used to communicate with servers. “oDoa0quieSeir6goowai7f”
<i>check</i>	The server to use for connection checks
<i>chipper</i>	The natural language processing server
<i>jdocs</i>	The remote JSON storage server
<i>logfiles</i>	The server to upload log files to
<i>tms</i>	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)

64. 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 below³⁶ gives the JSON keys, value format. It is implemented as gRPC/protobuf interaction over HTTP.

64.1. JDOCS INTERACTION

The JDoc message has the following fields

Field	Type	Description
<i>client_meta</i>	string	
<i>doc_version</i>	uint64	
<i>fmt_version</i>	uint64	
<i>json_doc</i>	string	

Table 283: JSON Parameters for JDoc request

64.2. DELETE DOCUMENT

64.2.1 Request

DeleteDocReq

- account
- thing – the thing id is a ‘vic:’ followed by the serial number
- doc_name

64.2.2 Response

DeleteDocResp

64.3. ECHO TEST

64.3.1 Request

EchoReq

- data

64.3.2 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.

64.4. READ DOCUMENTS

64.4.1 Request

ReadDocsReq

- account
- thing
- items

64.4.2 Response

ReadDocsResp

- items

64.5. READ DOCUMENT ITEM

64.5.1 Request

ReadDocsReq_Item

- doc_name
- my_doc_version

64.5.2 Response

ReadDocsResp_Item

- status (int)
- doc

64.6. WRITE DOCUMENT

64.6.1 Request

WriteDocReq

- account
- thing
- doc_name
- doc

64.6.2 Response

WriteDocResp

- status (int)
- latest_doc_version

65. NATURAL LANGUAGE PROCESSING

The “knowledge graph” Q&A server is done by Sound Hound.

66. 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:

HTTP header	Description
<i>Anki-App-Key:</i>	The appKey from the server configuration file.
<i>Usr-RobotESN:</i>	Vector’s serial number
<i>Usr-RobotOSRevision:</i>	The OS revision string from /etc/os-version-rev
<i>Usr-RobotOSVersion:</i>	The OS version string from /etc/os-version
<i>Usr-RobotRevision:</i>	The Anki revision string from /anki/etc/revision
<i>Usr-RobotTimestamp:</i>	The time of Vector’s internal clock.
<i>Usr-RobotVersion:</i>	The Anki version string from /anki/etc/version
<i>Usr-Username:</i>	

Table 284: Log upload
HTTP header fields

Note: the log uploader does not appear to be enabled in the production software.

67. 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:

Form fields	Description
<i>attachment_messages.log</i>	The “.log” file associated with the minidump. This is optional; only included if /run/das_allow_upload exists
<i>hostname</i>	<code> \${hostname}</code>
<i>robot.esn</i>	Vector’s serial number
<i>robot.os_version</i>	The OS version string from /etc/os-version
<i>robot.anki_version</i>	The Anki version string from /anki/etc/version
<i>upload_file</i>	The minidump “.dmp” file

Table 285: Crash
upload form fields

68. 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:

Keys	Value
<i>Action</i>	SendMessage
<i>MessageAttribute.1.Name</i>	DAS-Transport-Version
<i>MessageAttribute.1.Value.DataType</i>	Number
<i>MessageAttribute.1.Value.StringValue</i>	2

Table 286: DAS
Manager SQS key-value
pairs

<i>MessageAttribute.2.Name</i>	Content-Encoding
<i>MessageAttribute.2.Value.DataType</i>	String
<i>MessageAttribute.2.Value.StringValue</i>	gzip, base64
<i>MessageAttribute.3.Name</i>	Content-Type
<i>MessageAttribute.3.Value.DataType</i>	String
<i>MessageAttribute.3.Value.StringValue</i>	application/vnd.anki.json; format=normal; product=vic
<i>MessageBody</i>	
<i>Version</i>	2012-11-05 ³⁷

Note: there may be a body of compressed JSON data. These values are hardcoded in vic-dasmgr and libcozmo_engine

69. 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.

PART IV

Advanced Functions

This part describes items that are Vector's primary function.

- IMAGE PROCESSING. Vector vision system is sophisticated, with the ability to recognize marker, faces, and objects; to take photographs, and acts as a key part of the navigation system.
- MAPPING, NAVIGATION. A look at Vector's mapping and navigation systems
- ACCESSORIES. A look at Vector's home (charging station), companion cube and custom objects.



Steph Dere

drawing by Steph Dere

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

CHAPTER 16

Image Processing

This chapter describes Vector's 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), odometry and “simultaneous localization and mapping”

70. 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 data from the camera passes to device drivers, then to a separate daemon service and eventually passes to Vic-engine for the processing:

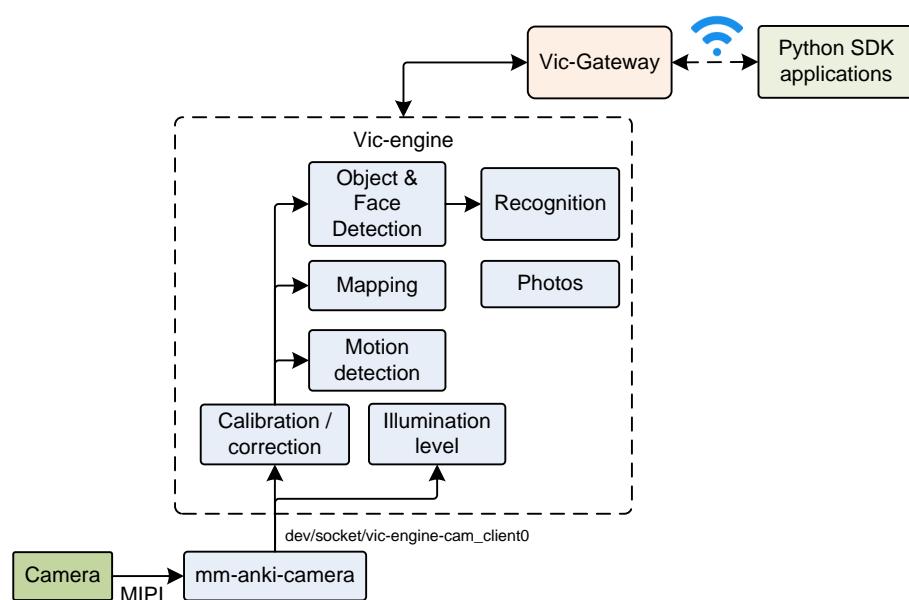


Figure 41: The camera architecture

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 area as fixed & unmovable.
- Faces
- Hands
- Pets (feature not completed)
- Other objects, like fruit, etc. (feature not completed)
- LASER pointers (feature not completed)

70.1. 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 $\sim 120^\circ$ 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.

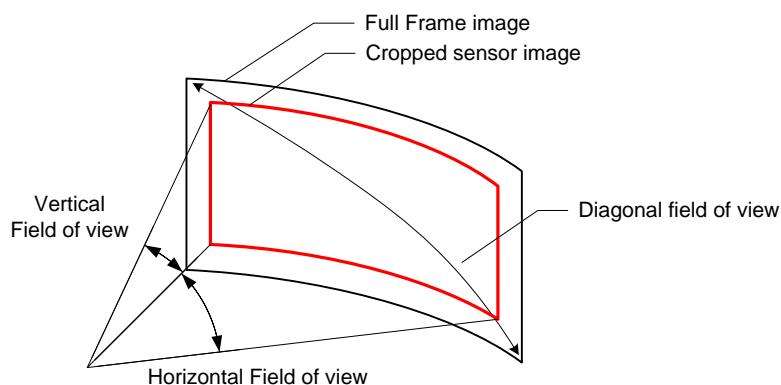


Figure 42: The camera field of view

Vector's calibration uses focal length instead of field of view. The two values are related:

$$\text{fieldOfView} w = 2 \arctan \frac{\text{sensor size}}{2 \text{focalLength}}$$

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
<i>cx</i>	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."
<i>cy</i>	float	
<i>distortionCoeffs</i>	float[]	
<i>fx</i>	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."
<ify< i=""></ify<>	float	
<i>ncols</i>	int	The width of the image in pixels. The value given is 640.
<i>nrows</i>	int	The height of the image in pixels. The value given is 360
<i>skew</i>	float	

Table 287: 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: The width and height of the camera here – 640x360 – make for an image that is quarter that of the 1280x720 value publically advertised. The most likely explanation is that the camera is calibrated (and typically used with) and used at a quarter of its full capacity. This was the case with Cozmo. Image processing tasks don't need more pixels, but take much, much longer (and more power) to process larger frames. First there is the added time to process each of the added pixels. Second, neural-net models (used for handle, pet and object recognition) are much larger as well, taking much longer to process.

The photographs were taken with less than the full resolution. It isn't known if Anki intended to eventually take photographs at a higher resolution.

70.2. 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
<i>AutoExposure</i>		
<i>BuildingOverheadMap</i>		
<i>CompositingImages</i>		
<i>ComputingStatistics</i>		
<i>CroppedFaceDetection</i>	DetectingFaces	Used to detect faces that are obscured or partly out of view.

Table 288: The Vision processes

<i>CyclingExposure</i>	AutoExposure
<i>DetectingBlinkAmount</i>	
<i>DetectingBrightColors</i>	
<i>DetectingFaces</i>	Used for face detection, and to trigger facial identification.
<i>DetectingGaze</i>	Detects the gaze and looks deep into their eyes with wonder and the hope of biscuits.
<i>DetectingHands</i>	Used to detect hands (for purposes of pouncing on them).
<i>DetectingIllumination</i>	
<i>DetectingLaserPoints</i>	
<i>DetectingMarkers</i>	Detects Vector's special square marker symbols.
<i>DetectingMotion</i>	
<i>DetectingOverheadEdges</i>	
<i>DetectingPeople</i>	
<i>DetectingPets</i>	
<i>DetectingSmileAmount</i>	
<i>DisableMarkerDetection</i>	
<i>EstimatingFacialExpression</i>	
<i>FullFrameMarkerDetection</i>	DetectingMarkers This part of the process of detecting Vector's special square marker symbols.
<i>FullHeightMarkerDetection</i>	DetectingMarkers This part of the process of detecting Vector's special square marker symbols.
<i>ImageViz</i>	
<i>FullWidthMarkerDetection</i>	DetectingMarkers This part of the process of detecting Vector's special square marker symbols.
<i>MarkerDetectionWhileRotatingFast</i>	DetectingMarkers This part of the process of detecting Vector's special square marker symbols.
<i>MeteringFromChargerOnly</i>	AutoExposure
<i>MinGainAutoExposure</i>	AutoExposure
<i>MirrorMode</i>	Displays the camera image on the LCD
<i>SavingImages</i>	

Notes: has rate control on processing those

70.3. 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.)

71. 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: Anki³⁹

³⁹ <https://forums.anki.com/t/camera-matrix-for-3d-positionning/13254/5>

72. 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.

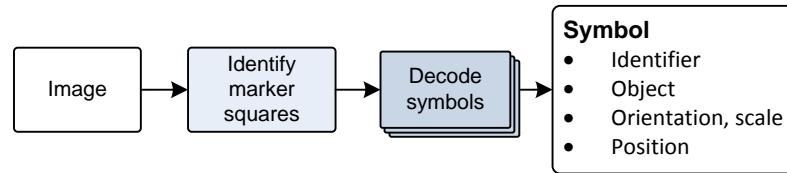


Figure 43: The processing of the image for symbols and objects

A key characteristic of the markers is a big, bold square line around it:

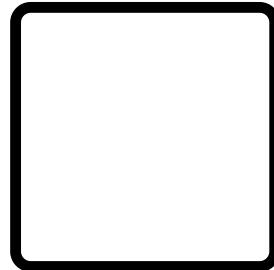


Figure 44: A typical rectangle around the visual markers

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.

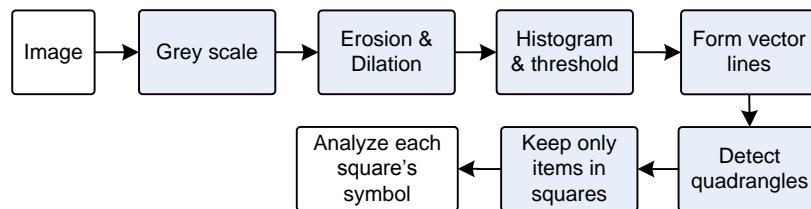


Figure 45: Preparing image for scanning for symbols

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 its size, and affine transform – how it is tilted up-and-down, and tilted away from the camera.

5. Screen the squares, tossing out those that are horribly distorted,
6. Analyze the pixels in the square to identify the code

72.1. 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 considered black (a 1 bit), and all others are white (0 bit).

72.2. 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 outermost 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 keep those that are big enough to analyze, and not distorted with a high skew or other asymmetries.

72.3. 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.

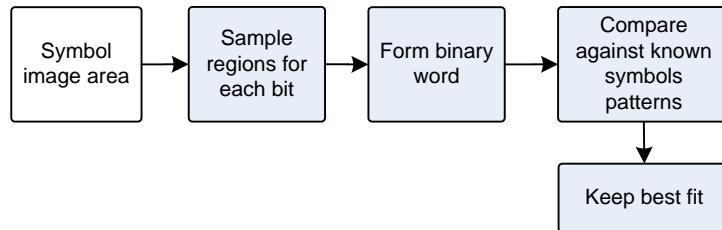


Figure 46: Decoding the symbol

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.

72.4. 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.

72.5. 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.

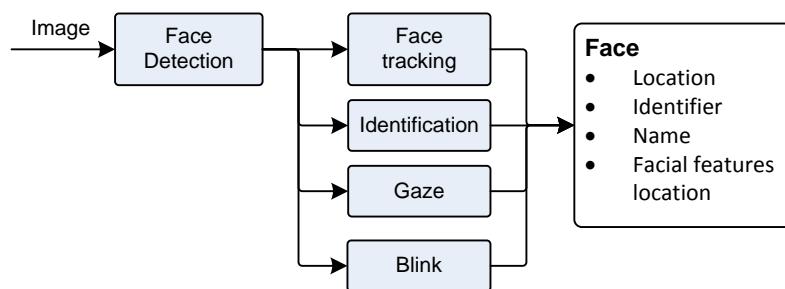
Vector knows (or is told) the physical size of the symbol, and the object holding the symbol.

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.

73. 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:

- 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.



Anki Cozmo SDK

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.)

73.1. FACE DETECTION

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).

73.2. 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.

73.3. 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 14 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

74. 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 Lite⁴⁰, an inference only neural-net discriminator.

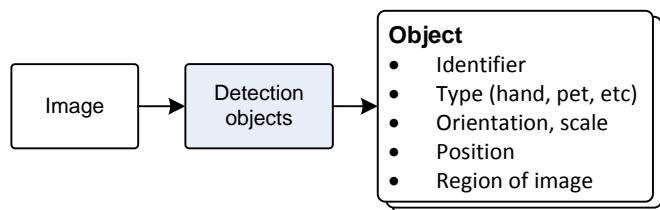


Figure 48: The processing of the image for symbols and objects

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.

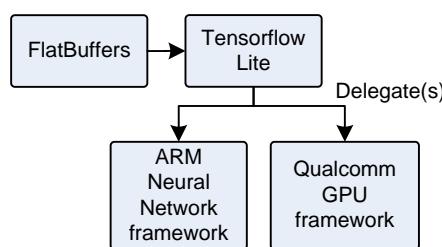


Figure 50: TensorFlow lite with hardware specific accelerators

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

75. 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.

Related to behaviour

75.1. COMMUNICATION INTERFACE

There are several commands to manage the photographs that Vector has taken. See Chapter 14 for more details.

- The *PhotoTaken* event is used to receive a notification when Vector has taken a photograph.
- 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

76. 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](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 Pattern 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 17

Mapping & Navigation

Vector builds an internal map to track where he can drive, where objects and faces are.

- Mapping Overview
- Navigation and Path Planning

77. MAPPING OVERVIEW

Vector tracks objects in two domains:

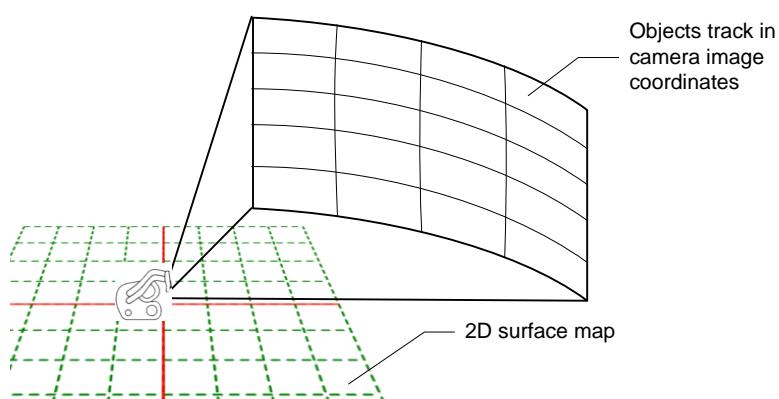


Figure 51: Mapping contexts

- A 2D map that is used to track where objects (especially objects whose marker symbols he recognizes), cliffs, and other things are on the surfaces that he can drive on. Vector uses this map to navigate. This map has an arbitrary origin and orientation.
- Vector also tracks where faces, pets 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's 2-D surface map system works with the localization and navigation subsystem. 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

78. MAP REPRESENTATION

Vector keeps track of the surface that he can drive on with a navigable 2D map. The map's 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.

78.1. 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 not.

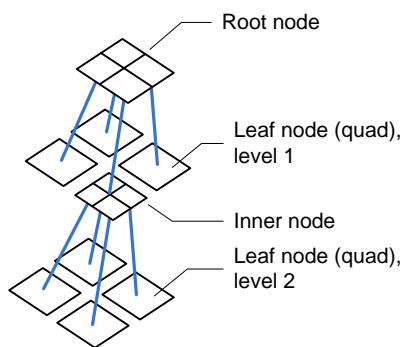


Figure 52: Structure of a quad-tree

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) and adds nodes down until it can contain the info at the edge of the map.

78.2. THE MAP'S STARTING POINT

Vector doesn't have a “north pole” or other global reference point to center his map on. When he powers on, or “whenever Vector is delocalized (i.e. whenever Vector no longer knows where he is – e.g. when he's picked up), Vector creates a new pose starting at (0,0,0) with no rotation. As Vector drives around, his pose (and the pose of other objects he observes – e.g. faces, his [cube], charger, etc.) is relative to this initial position and orientation.”

Client applications (the ones that talk via the HTTPS API) may also wish to know that the map was thrown out and a new one created – and thus know they should toss out their map and location of objects. Vector associates a unique identifier with each generation of the map called *origin_id*. Whenever a new map is created the “*origin_id* [is] incremented to show that [the] poses [of the

Anki SDK

map, Vector, and objects in the world] cannot be compared with earlier ones.” “Only poses of the same `origin_id` can safely be compared or operated on.”

78.3. 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.

79. BUILDING THE MAP

The map is made as Vector drives around – when he is on a mission, or just exploring. Each of the leaf quads (in the map) is associated with information about that space and what is contained there:

- What Vector knows is in the quad –a cliff, the edge of a line, an object with a marker symbol on it, or an object without a symbol (aka an obstacle),
- A list of what 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.

Vector subdivides quads to better represent the space. The quad probably is only slight bigger than the object in it. But the quad (probably) can be smaller than the object, to accommodate the object not oriented and aligned to fit quite perfectly in the quad. More than quad can refer to a contained object.

79.1. MAPPING CLIFFS AND EDGES

If a cliff (surface proximity) sensor has a large, significant change in value, Vector will make a note that there is a cliff sensor there. If the value has a smaller, but still noticeable change, he might make a note that there is a line edge there – an edge between a dark area and a light area.

79.2. TRACKING OBJECTS

Vector tracks objects (especially objects with markers) using the map, and other cross-referencing structures. Vector associates the following information with each object it tracks:

- The object's kind (dock, cube, etc)
- A pose. The image skew of the marker symbol gives some partial attitude (relative orientation) information about the object 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 objects position from 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.

If he sees a symbol, he uses the objects known size, the image scale, its pose (if known) and any time-of-flight information to (a) refine his estimated location on the map, (b) update the location and orientation of that object.

79.3. TIME OF FLIGHT

Vector has a time of flight sensor, pointing straight ahead. He can the sensor to better estimate the distance to the objects, barriers, open spots on the map, and to estimate his position. The sensor can be blocked by the arms, if they are in just the right lowered position; or by a cube that is being lifted. If the sensor is not blocked:

- Gathers samples of distances reported by the sensor,
- Apply a filter to them (probably a median filter), throwing out values that are too near or too far.
- Combining this with Vectors current position and orientation, and the distance to the object, he can estimate the objects position; and
- Vector can infer that the space between him and the object are free of other objects and obstacles. (This means splitting up the map quads into a fine-grained resolution along the narrow beam path.)

79.4. BUILDING A MAP WITH SLAM

Vector employs a mapping technique known as *simultaneous location and mapping* (SLAM) to integrate these (and other) steps. SLAM is a method to identify Vector's current position and orientation (relative to the map), and to construct that map.

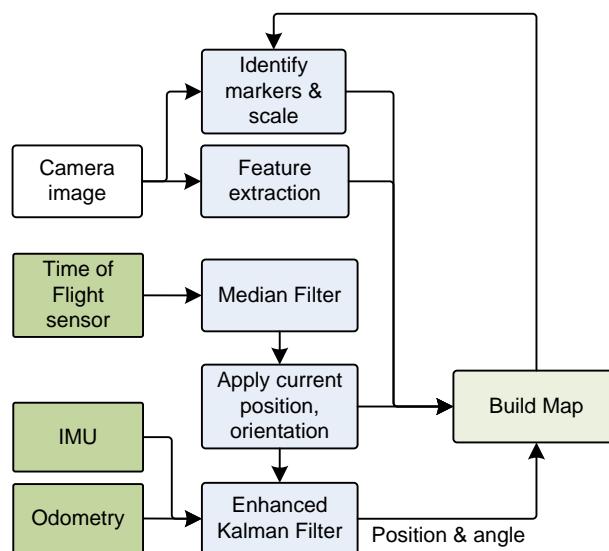


Figure 53: A typical localization and mapping functional block diagram

SLAM consists of multiple parts. It integrates the sensor for distance and movement. It also uses image processing to figure it out where it is at. It identifies landmarks, and information about them. In a sophisticated integration process, it can estimate Vector's orientation *and* if an object has moved. The estimate of Vectors orientation is based on turn information from the IMU, and refined by what it can see.

80. 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.

Vector’s path planning approach is unknown.

81. 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 18

Accessories

Vector's accessories include his charging station, companion cube, and custom items that can be defined thru the SDK.

- Accessories in general: symbols, docking
- Home & Charging Station
- Companion cube
- Custom items

82. ACCESSORIES IN GENERAL

Accessories have at least one maker symbol that Vector can recognize. Vector tracks the location and orientation based on this.

82.1. 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.

83. 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.

83.1. DOCKING

Vector's step in docking with the charging station are:

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.

84. 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 5.

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.

84.1. COMMUNICATION

Vector connects with the cube via Bluetooth LE. This communication link provides the ability for Vector to:

- Discover cubes
- Pair with a cube (note that Vector can pair with only one cube, and if he is not already paired, he will automatically pair with the first cube he receives Bluetooth LE advertising for.)
- Check the firmware version
- Update the cube firmware
- Check the cube’s battery level
- Detect the cube orientation
- Detect taps on the cube
- Turn the cubes lights on and off.

The HTTPS API provides the following commands of love:

- List the available cubes, see Chapter 14, *Cubes Available*
- Forget (or unpair) from his preferred cube, see Chapter 14, *Forget Preferred Cube*
- Pair to the first cube detected, see Chapter 14, *Set Preferred Cube*
- Connect to his cube , see Chapter 14, *Connect Cube*
- Disconnect from the cube, see Chapter 14, *Disconnect Cube*
- Dock with his cube, see Chapter 14, *Dock With Cube*
- Flash cube lights, see Chapter 14, *Flash Cube Lights* and *Set Cube Lights*. The later allows using a complex pattern.
- Pick up an object (his cube), see Chapter 14, *Pickup Object*
- Place his object (his cube) on the ground, see Chapter 14, *Place Object on Ground Here*
- Pop a wheelie, see Chapter 14, *Pop A Wheelie*
- Roll his cube, see Chapter 14, *Roll Block* and *Roll Object*

The state of the cube is reported to the HTTPS API.

As the state of the cube changes, the following events are posted to the API:

- The cubes battery level, see Chapter 14, *CubeBattery*
- A loss of the connection with the cube, see Chapter 14, *CubeConnectionLost*

- The robot state event (see Chapter 14, *RobotState*) provides other info about Vector's attempt to interact with the cube. This includes what object he is carrying. There are bits to indicate when
 - Vector is carrying his cube
 - His picking up or moving to dock with his cube
- The object event (see Chapter 14, *ObjectEvent*) provides other info about the state of the cube as it happens: taps, loss of connection, state of connection, being moved, etc.

84.2. CUBE ANIMATIONS

The cube light animation files are located in:

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

and within folders (and sub-folders) therein. This path is hard-coded into libcozmo-engine.

All of the cube light animation JSON files have the same structure. They are an array of structures. (There is usually one item, but there may be more.) Each structure may contain the following fields:

Field	Type	Units	Description	Table 289: The Cube LEDs JSON structure
<i>canBeOverridden</i>			<i>Optional.</i> Default is true.	
<i>duration_ms</i>	float	ms	if zero, do this until told to stop, otherwise perform the animation for this period and proceed to next structure or stop.	
<i>pattern</i>		<i>see below</i>	A structure describing the light patterns. Described below.	
<i>patternDebugName</i>	string		A text name that is associated with this structure. <i>Optional.</i>	

The pattern structure contains the following fields:

Field	Type	Units	Description
<code>offColors</code>	array of 4 colors	<i>RGBA</i>	Each color corresponds to each of the 4 cube lights. Each color is represented as an array of 4 integers (red, green, blue, and alpha), in the range 0..255. Alpha is always 255.
<code>offPeriod_ms</code>	<code>float[4]</code>	<i>ms</i>	The “off” duration for each of the 4 cube lights. This is the duration to show each cube light in its corresponding “off” color (in <code>offColors</code>).
<code>offset</code>	<code>float[4]</code>		always 0
<code>onColors</code>	array of 4 colors	<i>RGBA</i>	Each color corresponds to each of the 4 cube lights. Each color is represented as an array of 4 integers (red, green, blue, and alpha), in the range 0..255. Alpha is always 255.
<code>onPeriod_ms</code>	<code>float[4]</code>	<i>ms</i>	The “on” duration for each of the 4 cube lights. This is the duration to show each cube light in its corresponding “on” color (in <code>onColors</code>).
<code>rotate</code>	boolean		? Possibly to have the colors be assigned to the next clockwise (or counterclockwise) light periodically?
<code>transitionOffPeriod_ms</code>	<code>float[4]</code>	<i>ms</i>	The time to transition from the on color to the off color.
<code>transitionOnPeriod_ms</code>	<code>float[4]</code>	<i>ms</i>	The time to transition from the off color to the on color.

Table 290: The Cube LEDs pattern structure

This structure is very similar to that used by the backpack lights. See Chapter 9, *Backpack lights control*. The obvious differences are:

- There are 4 lights (instead) of three,
- The RGBA value range is 0..255; and
- There is a boolean “rotate” field.

85. 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.

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

PART V

Animation

Vector uses animations – “sequence[s] of highly coordinated movements, faces, lights, and sounds” – “to demonstrate an emotion or reaction.” This part describes how the animation system works.

- ANIMATIONS. An overview how Vector’s scripted animations represents the “movements, faces, lights and sounds;” and how they are coordinated
- DISPLAY & PROCEDURAL FACE. Vector displays a face to convey his mood and helps form an emotional connection with his human.
- AUDIO PRODUCTION. A look at how Vector’s sound effects and how he speaks
- MOTOR CONTROL. A look at how Vector’s moves.
- ANIMATIONS FILE FORMAT. The format of Vector’s binary animation file.



drawing by Steph Dere

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

CHAPTER 19

Animation

This chapter describes Vector's animation engine:

- Animation Engine
- Animation file formats

86. 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.”

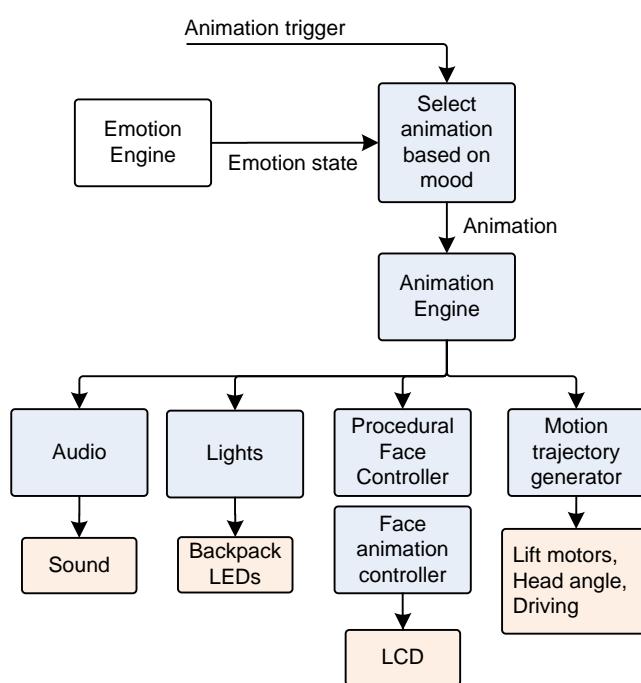


Figure 54: The behaviour-animation flow

86.1. ANIMATION TRACKS

Each animation may use one more tracks. The tracks can control:

- The display graphics – compositing a movie , text, and eyes layers
- Animating the backpack and cube lights

- Audio effects
- Moving the head
- Moving the lift
- Moving the treads, or navigating

When an animation is played, it locks the tracks that it is using, for the duration of the animation. If one of the tracks that it needs to use is already locked, the animation can't play (and generates an internal error).

When an animation is submitted to be played, a several of tracks (the lift, head and body) can be flagged to be ignored if they already used elsewhere by animation system.

87. ANIMATION FILES

The tables of animation groups are located in the following folder:

`/anki/data/assets/cozmo_resources/assets/animationGroups`

Inside are folders (grouping the animation groups), each of which holds JSON files that given some selection criteria. This path is hardcoded into `libcozmo_engine`.

The list of animations is located in

`/anki/data/assets/cozmo_resources/assets/sprites/anim_manifest.json`

This path is hardcoded into `libcozmo_engine`.

The animations binary files are held in the following folder:

`/anki/data/assets/cozmo_resources/assets/animations`

This path is hardcoded into `vic-anim`.

The sprite sequences – where a PNG makes up each frame of an animation – are held in folders within:

`/anki/data/assets/cozmo_resources/assets/sprites/spriteSequences`

and

`/anki/data/assets/cozmo_resources/config/devOnlySprites/spriteSequences`

These paths are hardcoded into `vic-anim`.

The screen layouts (and movement) for compositing visual animations of information is located in:

`/anki/data/assets/cozmo_resources/assets/compositeImageResources/imageLayouts`

`/anki/data/assets/cozmo_resources/assets/compositeImageResources/imageMaps`

These layouts are used by behaviors to construct what is displayed. These paths are hardcoded into `libcozmo_engine`.

88. 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.”

CHAPTER 20

Video Display & Face

Vector's LCD is used to display his face, which conveys his mood and forms an emotional connection with his human, and to display the results of behaviour interactions:

- The image compositor
- The sprite manager
- Animated compositions
- Procedural face

89. OVERVIEW OF THE DISPLAY

The TBD displays imagery – often moving imagery – on Vector's display. Drawn on screen

- 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” is 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.

89.1. 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.

90. RENDERING SYSTEM

The display is layered, placing sprites on top, followed by the compositional layout and, finally, the procedural face.

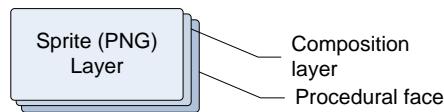


Figure 55: The display layers

The rendering and compositing these different layers look like:

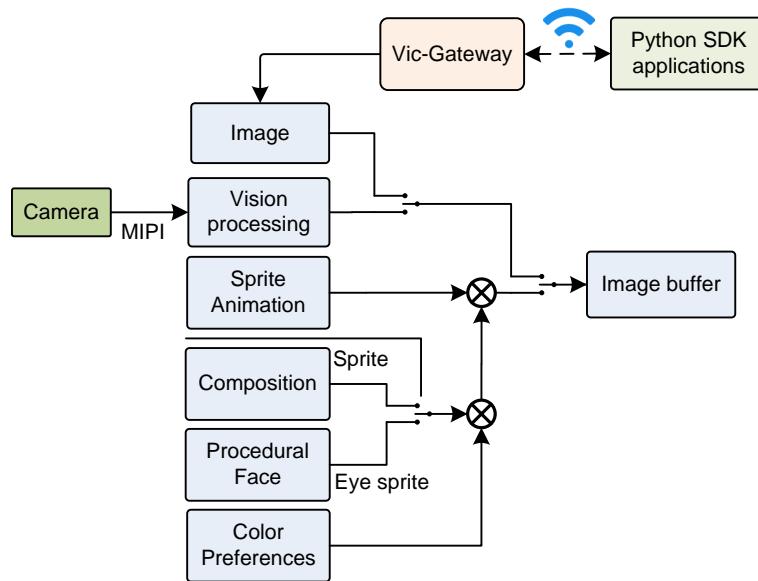


Figure 56: The display animations composition

As a functional flow, the top level is:

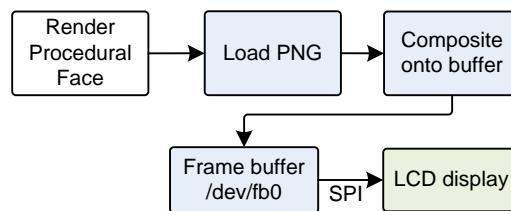


Figure 57: The display animations functional flow

90.1. FULL-SCREEN SPRITES

Full-screen sprites animations fully cover the display. These sprites can be displayed as a sequence. Each sprite is a PNG. Sprite sequences have the frame number appended to the name, and 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.

90.2. COMPOSITION SYSTEM

The screen is composed 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. The composition scripts include steps that can:

- Move the sprite box,
- Resizes the sprite box
- Changes the sprite in the box

91. PROCEDURAL FACE

Vector's dynamic, moving eyes are *the* gateway for an emotion connection. These eyes are drawn by the procedural face manager. The parameters of the face controls are divided into the overall view of the face and the individual characteristics of each eye:

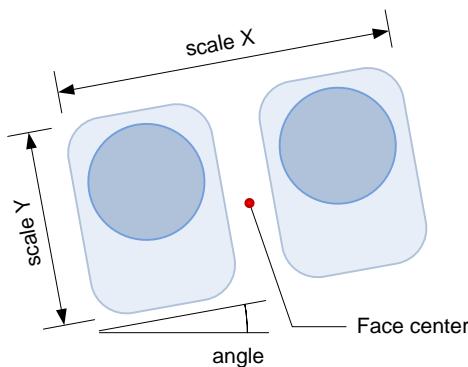


Figure 58: Face control points #1

The high level face animation parameters include:

- The color to draw the eyes in. Vector's eye color is a preference setting, but can be temporarily overridden by the SDK.
- The position of the center of the face
- The angle of the face; tilt (or rotation) of the face gives the impression of tilting the head
- The scaling of the height and width of the face
- There is a “scan line opacity” factor. This controls how much alternating lines are illuminated and darkened. A value of 1.0 has odd and even lines with the same coloring.

91.1. THE RENDERING OF INDIVIDUAL EYES

Each eye has individual animation parameters. These create a soft-face feel, by using a soft glow of the eye, along with rounded eyelids and cheeks that mask off some of the eye pixels.

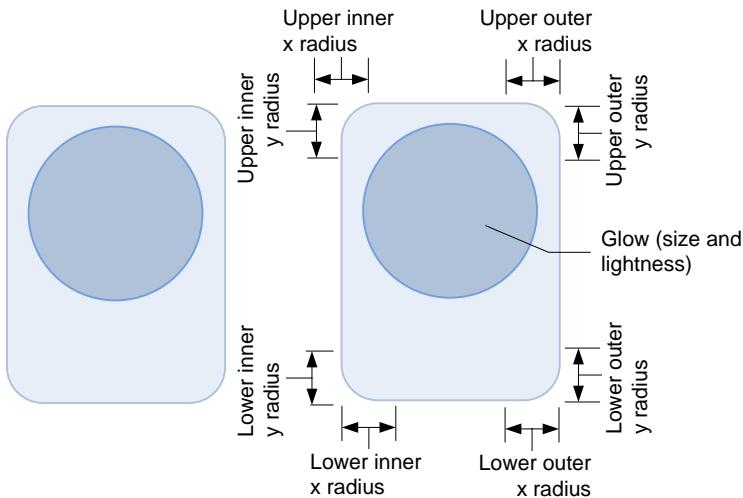


Figure 59: Basic parameters of an individual eye control

- Each eye has a gradient that gives it a soft spherical rounding effect. This is referred to as the “glow.” Its position helps create the illusion of gaze – that Vector is looking at something.
- The basic shape of the eye is controlled by the roundedness of the corners.
- The position of each eye is controlled by the center of the face;
- The size and width of the eye is created by the face’s scaling factors

Each eye has controls for its eyelids (or cheek, depending on your perspective):

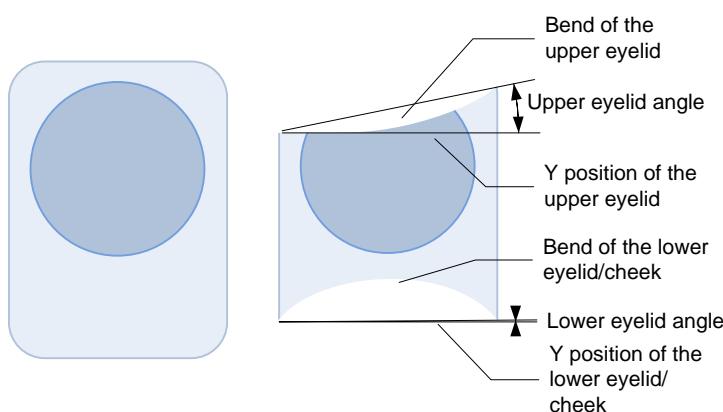


Figure 60: Parameters of an eye's eyelids (or cheeks)

- An arc represents the upper eyelid and erases (or occludes) the upper portion of the eyes; these help create the sleepy, frustrated/angry emotions.
- An arc represents the lower eyelid and cheek, and erases (or occludes) the lower portion of the eyes; these help create the happy emotions

An eye can be made smaller by having no bend to the eyelids, but moving the eyelids position closer to the center.

91.2. THE PROCESS OF DRAWING THE PROCEDURAL FACE

The process of drawing the procedural face is then:

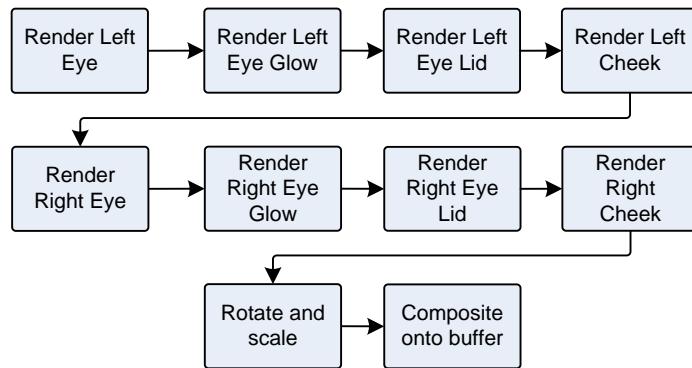


Figure 61: The functional flow of drawing the procedural face

92. COMMANDS

The HTTPS SDK API (Chapter 14) includes commands that affect the display

- see *Display Image RGB*
- see *Enable Mirror Mode*

CHAPTER 21

Audio Out

This chapter describes the sound output system:

- The audio output
- Text to speech
- Audio Effects

93. 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:

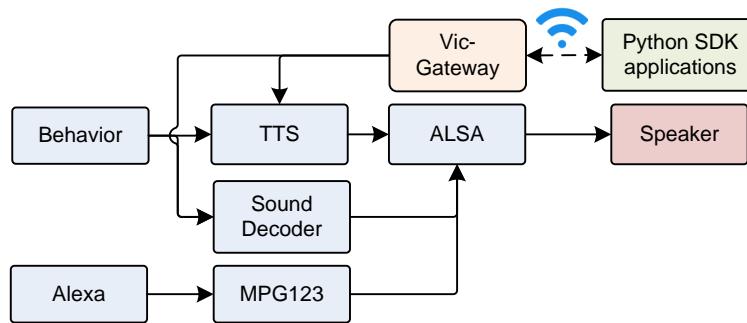


Figure 62: The audio output architecture

Doesn't seem to do sound effect and TTS at the same time

93.1. 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.

93.2. VOLUME CONTROL

Vector's volume can be set as a setting using the `UpdateSettings` command (see Chapter 14) and the `RobotSettingsConfig` structure (see chapter 25), or using the `Master Volume` command in Chapter 14. Note: the volume levels using settings doesn't fully match those in the master volume command.

93.3. 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

93.3.1 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:

Table 291: The JSON structure

Field	Type	Description & Notes
<code>pitch</code>	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.”
<code>shaping</code>	optional	
<code>speed</code>	float	
<code>speedTraits</code>	<code>speedTraits[]</code>	Optional array speed traits structures (see below)
<code>voice</code>	string	a path to the ini file within the [assets/cozmo_resources/tts] folder

Each `speedTraits` structure has the following fields:

Table 292: The `speedTraits` JSON structure

Field	Type	Description & Notes
<code>rangeMax</code>		
<code>rangeMin</code>		
<code>textLengthMax</code>		
<code>textLengthMin</code>		

⁴² The other OS key is “osx” which suggests that Vector’s software was development on an OS X platform.

93.3.2 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:

Field	Type	Description & Notes
<i>smartling</i>	structure	see below to the structure below. Note all smarting structures examined are the same.

Table 293: The JSON structure

The dictionary also includes keys, such as “BehaviorDisplayWeather.Rain” that map to a locale specific string. These have the following fields:

Field	Type	Description & Notes
<i>translation</i>	string	The text in the given locale. The string may have placeholders, such as {0}, where text is substituted in.

Table 294: The JSON structure

Each smartling structure has the following fields:

Field	Type	Description & Notes
<i>placeholder_format_custom</i>	array of strings	An array of patterns that represent possible placeholder patterns.
<i>source_key_paths</i>	array of strings	“/{*}” Strings are path of a JSON key?
<i>translate_paths</i>	array of strings	“*/translation” Strings are path of a JSON key?
<i>translate_mode</i>	string	“custom”
<i>variants_enabled</i>	boolean	

Table 295: The smartling JSON structure

This is handled by libcozmo_engine, including the key strings.

93.3.2.1 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

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.)

Table 296: The JSON structure

93.3.3 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.

94. PARAMETRIC 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.

95. COMMANDS

The HTTPS SDK API (Chapter 14) includes commands that affect the sounds

- Audio stream commands
- An external application can direct Vector to speak using the *Say Text* command. The response(s) provide status of where Vector is in the speaking process.
- *Master Volume* control

Note: can also trigger animations which play sounds effects as well.

96. 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 22

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

97. 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 chapter 19

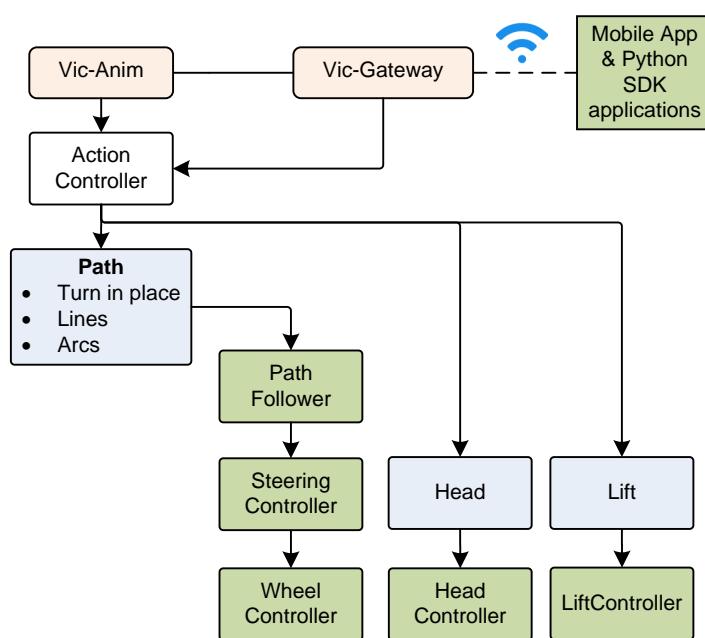


Figure 63: Motion controller

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.

97.1. 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.

97.2. 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.)

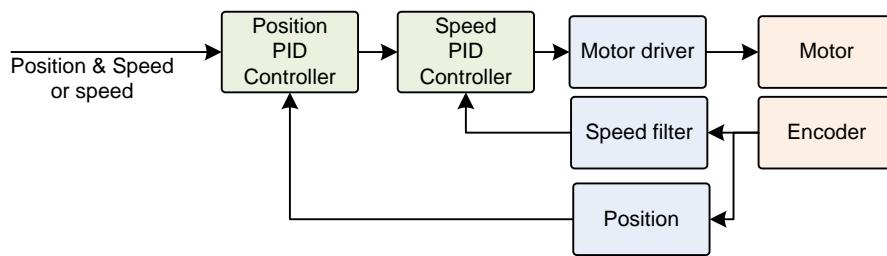


Figure 64: A typical motor controller

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 the shaft has turned – can be computed by counting the quadrature encoder events. These not only indicate when the motor has turned by a step, but also provide the direction the motor turned.

The speed of rotation is also computed, indirectly, from the encoder count. The typical approach is to take a derivative of the position, and filter it. Since the encoder is discrete, at slow speeds its update rate will produce false measures of shaft speed.

97.3. 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:

$$v_{left} = \omega(radius + \frac{1}{2}width)$$
$$v_{right} = \omega(radius - \frac{1}{2}width)$$

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:

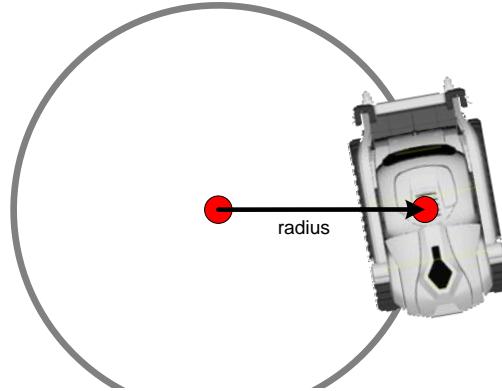


Figure 65: Radius of arc measurement

97.3.1 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.

98. MOTION CONTROL COMMANDS

The HTTPS SDK API (Chapter 14) 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*
- *Stop All Motors*
- *Turn In Place*
- *Set Head Angle*
- *Set Lift Height*
- *Go to Pose*
- *Turn Towards face*
- *Go To Object*

CHAPTER 23

Animation File Format

The binary animation files are the most significant of Vector's animation files. The file format provides for coordinating the display, motion, and sound responses.

- Animation file format overview
- Structures in the file

99. ANIMATION BINARY FILE FORMAT

The schema and format of the animation binary file is given as a flatbuffer specification. This specification is located at:

`/anki/data/assets/cozmo_resources/config/cozmo_anim.fbs`

Note: this file is not read by any program in Vector. A compatible parser is compiled in.

99.1. OVERVIEW OF THE FILE FORMAT

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.

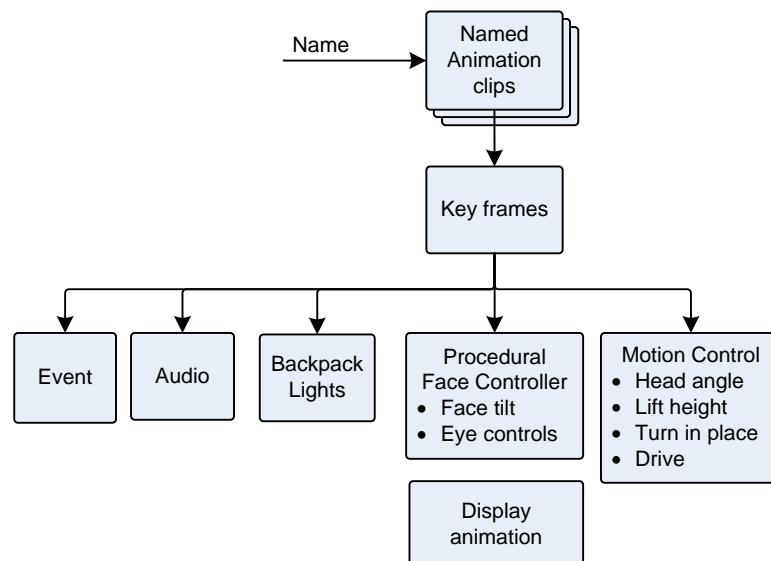


Figure 66: The animation file structure

Each of the tracks within the clip is composed of key frames (with settings for each of the relevant tracks) that are triggered at different points in time.

99.2. 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. The key differences between Cozmo and Vector's format is that Vector includes audio tracks, plus some minor extra fields, and fewer backlight LEDs.

The PyCozmo project has the (experimental) ability extract Cozmo's animations, and may be useful for this transcoding.

100. STRUCTURES

The animation file starts with an AnimClips structure. Unless specified otherwise, these are the same as in Cozmo.

100.1. 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:

Field	Type	Description
<i>clips</i>	AnimClip[]	An array of animation clips

Table 297: AnimClips structure

100.2. ANIMCLIP

The AnimClip has the following fields:

Field	Type	Description
<i>Name</i>	string	The name of the animation clip
<i>keyFrames</i>	KeyFrames	The key frames for each of the tracks for this animation clip

Table 298: AnimClip structure

100.3. AUDIOEVENTGROUP

The AudioEventGroup structure is new to Vector. This structure has the following fields:

Field	Type	Units	Description
<i>eventIds</i>	uint[]		
<i>volumes</i>	float[]		
<i>probabilities</i>	float[]		

Table 299: AudioEventGroup structure

100.4. AUDIOPARAMETER

The AudioParameter structure is new to Vector. This structure has the following fields:

Field	Type	Units	Description	Table 300: AudioParameter structure
<i>parameterId</i>	uint		default: 0	
<i>value</i>	float		default: 0	
<i>time_ms</i>	uint	ms	default: 0	
<i>curveType</i>	ubyte		default: 0	

100.5. AUDIOSTATE

The AudioState structure is new to Vector. This structure has the following fields:

Field	Type	Description	Table 301: AudioState structure
<i>switchGroupId</i>	uint	default: 0	
<i>stateId</i>	uint	default: 0	

100.6. AUDIOSWITCH

The AudioSwitch structure is new to Vector. This structure has the following fields:

Field	Type	Description	Table 302: AudioSwitch structure
<i>switchGroupId</i>	uint	default: 0	
<i>stateId</i>	uint	default: 0	

100.7. BACKPACKLIGHTS

The BackpackLights structure is used to animate the LEDs on Vector's back. This structure has the following fields:

Field	Type	Units	Description	Table 303: BackpackLights structure
<i>triggerTime_ms</i>	uint	ms	The time at which the backlights are triggered.	
<i>durationTime_ms</i>	uint	ms	How long the animation lasts	
<i>Front</i>	float[]			
<i>Middle</i>	float[]			
<i>Back</i>	float[]			

see also: Chapter 9 *Backpack lights control*

Note: Cozmo's animation structure includes a left and right LED animation.

100.8. BODYMOTION

The BodyMotion structure is used to specify driving motions for Vector. This structure has the following fields:

Field	Type	Units	Description
<i>triggerTime_ms</i>	uint	ms	The time at which the motion is triggered.
<i>durationTime_ms</i>	uint	ms	How long the animation lasts
<i>radius_mm</i>	string	mm	TODO: is this the name of a key emotion thing? “TURN_IN_PLACE”, “STRAIGHT”, possibly a value?
<i>speed</i>	short		

Table 304: BodyMotion structure

100.9. EVENT

The Event structure has the following fields:

Field	Type	Units	Description
<i>triggerTime_ms</i>	uint	ms	The time at which the motion is triggered.
<i>event_id</i>	string		

Table 305: Event structure

100.10. FACEANIMATION

The FaceAnimation structure is used to specify the JSON file to animation Vector’s display. This structure has the following fields:

Field	Type	Units	Description
<i>triggerTime_ms</i>	uint	ms	The time at which the motion is triggered.
<i>animName</i>	string		
<i>scanlineOpacity</i>	float		This is new for Vector. Default: 1.0

Table 306: FaceAnimation structure

100.11. HEADANGLE

The HeadAngle structure is used to specify how to move Vector’s head. This structure has the following fields:

Field	Type	Units	Description
<i>triggerTime_ms</i>	uint	ms	The time at which the motion is triggered.
<i>durationTime_ms</i>	uint	ms	How long the animation lasts
<i>angle_deg</i>	ubyte	deg	The angle to move the head to
<i>angleVariability_deg</i>	ubyte	deg	default: 0

Table 307: HeadAngle structure

100.12. LIFTHEIGHT

The LiftHeight structure is used to specify how to move Vector's lift. This structure has the following fields:

Field	Type	Units	Description
<i>triggerTime_ms</i>	uint	ms	The time at which the motion is triggered.
<i>durationTime_ms</i>	uint	ms	How long the animation lasts
<i>height_mm</i>	ubyte	mm	The height to lift the arms to
<i>heightVariability_mm</i>	ubyte	mm	default: 0

Table 308: LiftHeight structure

100.13. KEYFRAMES

The KeyFrames structure the following fields:

Field	Type	Description
<i>LiftHeightKeyFrame</i>	LiftHeight[]	
<i>ProceduralFaceKeyFrame</i>	ProceduralFace[]	
<i>HeadAngleKeyFrame</i>	HeadAngle[]	
<i>RobotAudioKeyFrame</i>	RobotAudio[]	
<i>BackpackLightsKeyFrame</i>	BackpackLights[]	
<i>FaceAnimationKeyFrame</i>	FaceAnimation[]	
<i>EventKeyFrame</i>	Event[]	
<i>BodyMotionKeyFrame</i>	BodyMotion[]	
<i>RecordHeadingKeyFrame</i>	RecordHeading[]	
<i>TurnToRecordedHeadingKeyFrame</i>	TurnToRecordedHeading[]	

Table 309: KeyFrames structure

100.14. PROCEDURALFACE

The ProceduralFace structure is used squash, stretch and shake Vectors face in cartoonish ways. It has the following fields:

Field	Type	Units	Description
<i>triggerTime_ms</i>	uint	ms	The time at which the motion is triggered.
<i>faceAngle</i>	float		default: 0
<i>faceCenterX</i>	float		default: 0
<i>faceCenterY</i>	float		default: 0
<i>faceScaleX</i>	float		default: 1.0
<i>faceScaleY</i>	float		default: 1.0
<i>leftEye</i>	float[]		If present, these describe modifications to the eye – lid, cheeks, and shape of the eye. They have the structure given below.
<i>rightEye</i>	float[]		If present, these describe modifications to the eye – lid, cheeks, and shape of the eye. They have the structure given below.
<i>scanlineOpacity</i>	float		This is new for Vector. default: 1.0

Table 310:
ProceduralFace
structure

The arrays of floats for each eye in animations for *Cozmo* have been deciphered. They are presumed to be the same for Vector:

PyCozmo

Field	Default	Description
<i>lower_inner_radius_x</i>	0.5	
<i>lower_inner_radius_y</i>	0.5	
<i>lower_outer_radius_x</i>	0.5	
<i>lower_outer_radius_y</i>	0.5	
<i>upper_inner_radius_x</i>	0.5	
<i>upper_inner_radius_y</i>	0.5	
<i>upper_outer_radius_x</i>	0.5	
<i>upper_outer_radius_y</i>	0.5	
<i>upper_lid_y</i>	0.0	The vertical position of the upper eye lid (which occludes the eye).
<i>upper_lid_angle</i>	0.0	The angle of the upper eye lid.
<i>upper_lid_bend</i>	0.0	The bend to the upper eye lid.
<i>lower_lid_y</i>	0.0	The vertical position of the lower eye lid / cheek (which occludes the eye).
<i>lower_lid_angle</i>	0.0	The angle of the lower eye lid / cheek.
<i>lower_lid_bend</i>	0.0	The bend to the lower eye lid / cheek.

100.15. RECORDHEADING

The RecordHeading structure has the following fields:

Field	Type	Units	Description
<i>triggerTime_ms</i>	uint	ms	The time at which the motion is triggered.

Table 311:
RecordHeading structure

100.16. ROBOTAUDIO

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
<i>triggerTime_ms</i>	uint	ms	The time at which the audio is triggered.
<i>eventGroups</i>	AudioEventGroup[]		
<i>state</i>	AudioState[]		
<i>switches</i>	AudioSwitch[]		
<i>parameters</i>	AudioParameter[]		

Table 312: *RobotAudio* structure

100.17. TURNTORECORDEDHEADING

The TurnToRecordedHeading structure has the following fields:

Field	Type	Units	Description
<i>triggerTime_ms</i>	uint	ms	The time at which the motion is triggered.
<i>durationTime_ms</i>	uint	ms	How long the animation lasts
<i>offset_deg</i>	short	deg	default: 0
<i>speed_degPerSec</i>	short	deg/sec	
<i>accel_degPerSec2</i>	short	deg/sec ²	default: 1000
<i>decel_degPerSec2</i>	short	deg/sec ²	default: 1000
<i>tolerance_deg</i>	ushort	deg	default: 2
<i>numHalfRevs</i>	ushort		default: 0
<i>useShortestDir</i>	bool		default: false

Table 313:
TurnToRecordedHeading structure

PART VI

High Level AI

This part describes items that are Vector's behaviour function.

- BEHAVIOR. A look at how Vector's moves.
- EMOTION MODEL. A look at how Vector vision system



Steph Dere

drawing by Steph Dere

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

CHAPTER 24

Behavior

This chapter describes Vector's action, behaviour, and emotion system:

- Actions and behaviour queues
- The emotion-behaviour system, and stimulation

101. OVERVIEW

How does Vector get excited for praise, and then decide to go exploring and play? How does he decide it's time to take a nap?

Vector's high-level AI – his emotions, sense of the environment and himself, and behaviours – are a key part of how he creates a compelling character. He has an emotional state that is seen in his affect – his facial expression, head and arm posture – how he behaves and responds, as well as the actions he initiates.

102. ACTIONS AND BEHAVIORS

Actions and “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.”

Anki SDK

102.1. ACTIONS AND THE ACTION QUEUES

Animations can be submitted with which tracks of the animation to disable. This allows multiple actions can be run at the same time. If the action requires a track that is already in use, the action isn't run, and returns an error. Actions can automatically retry if a problem was encountered.

Actions can have associated “tag” used to refer to that running instance. The client can cancel the action.

“For commands such as `go_to_pose`, `drive_on_charger` and `dock_with_cube`, Vector uses path planning, which refers to the problem of navigating the robot from point A to B without collisions. Vector loads known obstacles from his map, creates a path to navigate around those objects, and then starts following the path. If a new obstacle is found while following the path, a new plan may be created.”

Anki SDK

102.2. BEHAVIORS

Unlike actions, one behavior can be active at a time. The others are waiting in a stack or queue. A behavior is submitted with priority; if its priority is higher priority than the current one, it is run instead. The old behavior is pushed down in the stack. When a behavior completes, the next high priority one is resumed.

There is no provision to cancel a behavior.

103. EMOTIONS, AND STIMULATION

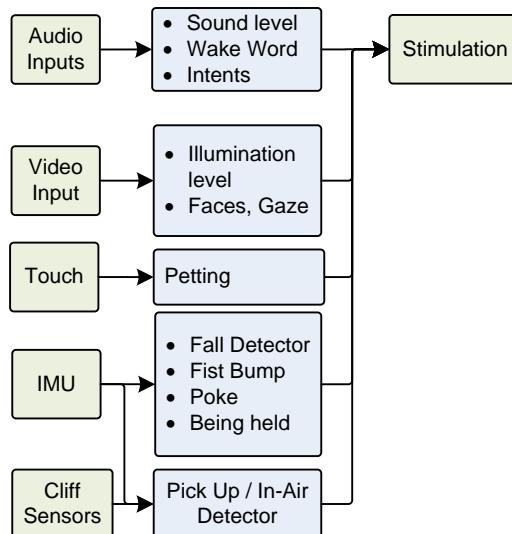
Emotions are how Vector responds to praise. Vector uses an emotional model to drive behavior and modify his responses. The name should be taken too seriously, as it doesn't model psychological moods, and other concepts. It does just enough to convey character.

103.1. STIMULATION

Vector uses a concept of a stimulation level to guide how much he should initiate

"When stimulation is low, the robot is chill,".. Vector is studiously observing but not acting out. "Then if you start making noise, or make eye contact with the robot, and certainly if you say 'Hey Vector,' that spikes [stimulation] way up..." But Vector also picks up subtler actions—peripheral movement and noises, for instance, or the room lights turning on and off. "If he gets stimulated enough, he'll drive off his charger and start to socialize with you, ... say your name, greet you, give you a fist-bump, potentially."

Captain 2018 quoting
Brad Neuman



103.2. THE EMOTION MODEL

Vector has just enough dimensions/aspects to his emotion model to drive responses/behaviours; and give a sense of personality. (The more dimensions used, the harder it is to get right, and the less convincing the character is.). Vector uses four dimensions to his emotional state:

- happy
- confident
- social
- stimulated

Each dimension is

the level to which he is stimulated, happy, social, and confident. Hearing his name stimulates Vector, for instance, but it also makes him more social.

Captain 2018

Vector's confidence is affected by his success in the real world. The hooks on his arms sometimes don't line up with those on his cube, for instance, and he can't pick it up.

Sometime he gets stuck while driving around. These failures make him feel less confident, while successes make him more confident and more happy.

The emotions and responses to stimuli are managed by the “MoodManager”

103.3. MOOD MANAGER

The configuration files for the mood manager are located in a folder at:

`/anki/data/assets/cozmo_resources/config/engine/emotionevents`

This is path hardcoded into libcozmo_engine. It is a folder that contains a set of JSON files, all with the same structure. Each of these files are loaded. The structure of the files is a structure containing the following fields:

Field	Type	Description	Table 314: CheckUpdateStatusResponse JSON structure
<i>emotionEvents</i>	EmotionEvent[]	An array emotion event structures (see below).	

Field	Type	Description	Table 315: EmotionEvent JSON structure
<i>name</i>	string	The name of the event (see table XXX)	
<i>emotionAffectors</i>	EmotionAffectord[]	The impact on the emotion state	
<i>repetitionPenalty</i>	TBD[]	<i>Optional.</i> This is an array of structures with “x” and “y” floats. x is seconds? y is in the range 0 to 1.0 {how fast it ramps up the stimulus? if the stimulus disappears in that time, it stops?}	

The emotion affector describes how the emotion axis should be modified.

Field	Type	Description	Table 316: EmotionAffectord JSON structure
<i>emotionType</i>	string	The type of emotion (“Happy”, “Confident”, “Stimulated” or “Social”)	
<i>value</i>	float	The value to add to the emotional state. The range is -1 to 1	

Altogether, the files respond to the following “emotion event” names. Some are external stimuli, some are events in general, some are events regarding whether or not a behaviour succeed, or failed (failed with retry, failed with abort)

	Emotion Name	Description and notes	Table 317: The emotion event names
<i>Ambient light</i>	ReactToDark		
<i>Charger</i>	DriveOffCharger		
	MountChargerSuccess		
	PlacedOnCharger		
<i>Cube</i>	CubeSpinner		
	KeepawayPounce		

	KeepawayStarted
	PickingOrPlacingActionFailedWithAbort
	PickingOrPlacingActionFailedWithRetry
	PickupSucceeded
	RollSucceeded
<i>Driving</i>	CliffReact
	DrivingActionFailedWithAbort
	DrivingActionFailedWithRetry
	DrivingActionSucceeded
	ExploringExamineObstacle
	FoundObservedObject
	ReactToObstacle
<i>Faces</i>	DrivingToFace
	DriveToFaceSuccess
	EnrolledNewFace
	EyeContactReaction
	GreetingSayName
	InteractWithFaceRetry
	InteractWithNamedFace
	InteractWithUnnamedFace
	LookAtFaceVerified
<i>Intents</i>	BeQuietVoiceCommand
	FistBumpLeftHanging
	FistBumpSuccess
	NoValidVoiceIntent
	OnboardingStarted
	ReactToTriggerWord
	RespondToGoodNight
	RespondToShortVoiceCommand
<i>Motion sensing</i>	ReactToMotion
	ReactToPickedUp
	ReactToUnexpectedMovement
	RobotShaken
<i>Power State</i>	Asleep
	Sleeping

<i>Petting</i>	PettingBlissLevelIncrease PettingReachedMaxBliss PettingStarted
<i>Sound</i>	DanceToTheBeat ReactToSoundAsleep ReactToSoundAwake
<i>Timer</i>	TimerRinging

104. REFERENCES & RESOURCES

Captain, Sean; *Can emotional AI make Anki's new robot into a lovable companion?* , Fast Company, 2018-8-8
<https://www.fastcompany.com/90179055/can-emotional-ai-make-ankis-new-robot-into-a-lovable-companion>

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

PART VII

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



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

CHAPTER 25

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

105. THE ARCHITECTURE

The architecture for setting and storing settings, statistics, account information is:

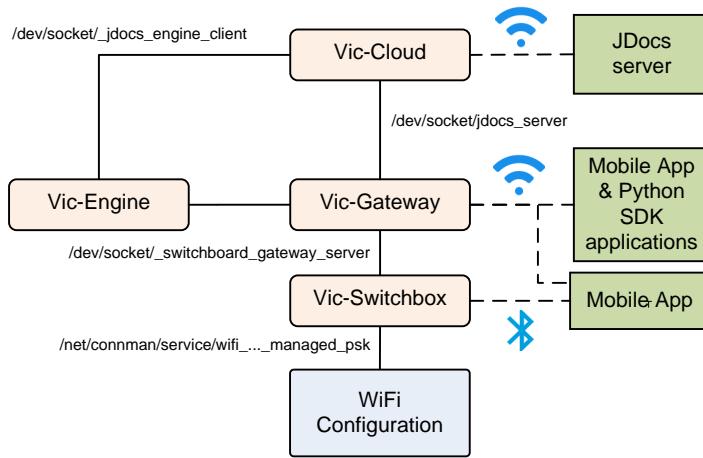


Figure 67: The architecture for storing preferences, account info, entitlements, and tracking stats

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.

106. 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

107. 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 318: The owners account information

JSON Key	units	Description & Notes
<i>user_id</i>	base64	A base64 token to identify the user
<i>created_by_app_name</i>	string	The name of the mobile application that register the owner. Example: “chewie”
<i>created_by_app_platform</i>	string	The mobile OS version string when the mobile application created the owners account. Example “ios 12.1.2; iPhone8,1
<i>created_by_app_version</i>	string	The version of the mobile application that register the owner. Example: “1.3.1”
<i>deactivation_reason</i>		
<i>dob</i>	YYYY-MM-DD	The owner’s date of birth (the one given at time of registration)
<i>drive_guest_id</i>	GUID	A GUID to identify the owner. This is the same as the “player_id”
<i>email</i>	string	The email address used to register the account; the same as the user name.
<i>email_failure_code</i>		The reason that the email was unable to be verified
<i>email_is_blocked</i>	boolean	
<i>email_is_verified</i>	boolean	True if the email verification has successfully completed. False otherwise.
<i>email_lang</i>	IETF language tag	The IETF language tag of the owner’s language preference. example: “en-US”
<i>family_name</i>	string	The surname of the owner; null if not set
<i>gender</i>	string	The gender of the owner; null if not set
<i>given_name</i>	string	The given of the owner; null if not set
<i>is_email_account</i>	boolean	
<i>no_autodelete</i>	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.

<i>password_is_complex</i>	boolean	
<i>player_id</i>	GUID	A GUID to identify the owner. This is the same as the “drive_guest_id”
<i>purge_reason</i>		
<i>status</i>	string	Example “active”
<i>time_created</i>	string	The time, in ISO8601 format, that the account was created
<i>user_id</i>	base64	A base64 token to identify the owner
<i>username</i>	string	Same as the email address

108. 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 14. 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.”).

108.1. ENUMERATIONS

108.1.1 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:

Name	Value	Description	
BUTTON_WAKEWORD_ALEXA	1	When the button is pressed, act as if “Alexa” was said.	
BUTTON_WAKEWORDHEY_VECTOR	0	When the button is pressed, act as if “Hey, Vector” was said.	

Table 319:
ButtonWakeWord
Enumeration

108.1.2 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:

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		

Table 320: EyeColor
Enumeration

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
<i>Hue</i>	float	The hue to use for the color
<i>Saturation</i>	float	The saturation to use for the color.

Table 321: 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.

108.1.3 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
<i>MUTE</i>	0	
<i>LOW</i>	1	
<i>MEDIUM_LOW</i>	2	
<i>MEDIUM</i>	3	
<i>MEDIUM_HIGH</i>	4	
<i>HIGH</i>	5	

Table 322: Volume Enumeration

108.2. ROBOTSETTINGSCONFIG

The RobotSettingsConfig structure has the following fields:

Field	Type	Description & Notes
<i>button_wakeword</i>	ButtonWakeWord	When the button is pressed, act as if this wake word (“Hey Vector” vs “Alexa”) was spoken. default: 0 (“Hey Vector”)
<i>clock_24_hour</i>	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. default: false
<i>default_location</i>	string	default: “San Francisco, California, United States”
<i>dist_is_metric</i>	boolean	If true, use metric units for distance measures; if false, use

Table 323: The RobotSettingsConfig JSON structure

		imperial units. default: false
<i>eye_color</i>	<code>EyeColor</code>	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.) default: 0 (<code>TIP_OVER_TEAL</code>)
<i>locale</i>	<code>strong</code>	The IETF language tag of the owner's language preference – American English, UK English, Australian English, German, French, Japanese, etc. default: "en-US"
<i>master_volume</i>	<code>Volume</code>	default: 4 (<code>MEDIUM_HIGH</code>)
<i>temp_is_fahrenheit</i>	<code>boolean</code>	If true, use Fahrenheit for temperature units; otherwise use Celsius. ⁴⁵ default: true
<i>time_zone</i>	<code>string</code>	The "tz database name" for time zone to use for the time and alarms. 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:

Field	Type	Description & Notes
<i>defaultValue</i>		The value to employ unless one has been given by the operator or other precedent.
<i>updateCloudOnChange</i>	<code>boolean</code>	true if the value is pushed to the colour when it is changed by the operator. False if not. Won't be restored..?

Table 324: The setting control structure

It is implied that the setting value is to be pulled from the Cloud when the robot is restored after clearing.

109. 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 324: The setting control structure*, used in settings in the previous section.

110. 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 324: The setting control structure*, used in settings in an earlier section.

The settings include:

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. default: “en-US”
DATA_COLLECTION	boolean	default: false

Table 325: The Cozmo account settings

111. 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 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.

111.1. 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:

Table 326: The feature flag structure

Field	Type	Description & Notes
<i>enabled</i>	boolean	True if the feature is enabled, false if not
<i>feature</i>	string	The name of the feature

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.

111.2. 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 *Get Feature Flag* (Chapter 14) query for more details.

112. 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 376: 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.

113. REFERENCES & RESOURCES

Wikipedia, *List of tz database time zones*,
https://en.wikipedia.org/wiki/List_of_tz_database_time_zones

CHAPTER 26

The Software Update process

This chapter describes the software update process

- The software architecture
- The software update process
- How to extract official program files

114. THE ARCHITECTURE

The architecture for updating Vector's software is:

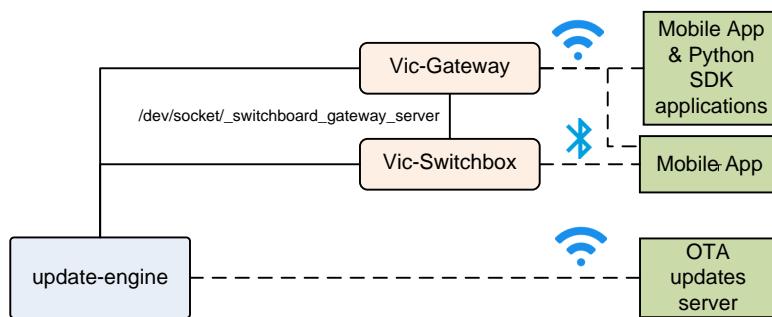


Figure 68: The architecture for updating Vector's software

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

115. 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.

115.1. MANIFEST.INI

The manifest.ini is checked by verifying its signature⁴⁶ 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

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:

Key	Description
<i>base_version</i>	The version that Vector must be at in order to accept this update. Honored only in delta updates.
<i>bytes</i>	The number of bytes in the uncompressed archive
<i>compression</i>	gz (for gzipped). This is the only supported compression type.
<i>delta</i>	1 if this is a delta update; 0 otherwise
<i>encryption</i>	1 if the archive file is encrypted; 0 if the archive file is not encrypted.
<i>sha256</i>	The digest of the decompressed file must match this
<i>wbits</i>	31. Not used by update-engine

Table 328:
manifest.ini image stream sections

115.2. HOW TO DECRYPT THE OTA UPDATE ARCHIVE FILES⁴⁷

How to decrypt the OTA update archive files:

```
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
```

Example 8: Decrypting the OTA update archives

To use OpenSSL 1.1.0 or later, add “-md md5” to the command:

```
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
```

Example 9: Decrypting the OTA update archives with Open SSL 1.1.0 and later

Note: the password on this file is insecure (ota.pas has only a few bytes⁴⁸) 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 7 *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/YIYQsX08OD4/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”

116. THE UPDATE PROCESS

116.1. STATUS DIRECTORY

The update-engine provides its status thru a set of files in the /run/update-engine folder.

File	Description
<i>done</i>	If this file exists, the update has completed
<i>error</i>	The error code representing why the update failed. See Appendix D, <i>Table 347: OTA update-engine status codes</i>
<i>expected-download-size</i>	The expected file size (the given total size of the OTA file) to download.
<i>expected-size</i>	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.
<i>progress</i>	Indicates how many of the bytes to download have been completed, or how much of the partitions have been written.

Table 329: update-engine status file

This folder also holds the unencrypted, uncompressed files from the OTA file:

- manifest.ini
- manifest.sha256
- delta.bin
- aboot.img
- boot.img

116.2. 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

117. RESOURCES & RESOURCES

<https://source.android.com/devices/bootloader/flashing-updating>
Describes the a/b process as it applies to android

CHAPTER 28

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

118. 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

118.1. THE SOFTWARE INVOLVED

There are many different programs and libraries used in the diagnostic and logging area. The table below summarizes of them:

Program / Library	Description
<i>animfail</i>	This is started by the <i>animfail</i> service.
<i>anki-crash-log</i>	Copies the last 500 system messages and the crash dump passed to the command line to a given log file. This is called by <i>vic-cloud</i> , <i>vic-dasmgr</i> , <i>vic-engine</i> , <i>vic-gateway</i> , <i>vic-log-kernel-panic</i> , <i>vic-log-upload</i> , <i>vic-robot</i> , <i>vic-switchboard</i> , and the <i>anki-crash-log</i> service.
<i>ankitrace</i>	This program wraps the Linux tract toolkit (LTTng). This program is not present in Vector’s file system. This is called by <i>fault-code-handler</i> .
<i>cti_vision_debayer</i>	This is not called.
<i>diagnostics-logger</i>	Bundles together several log and configuration states into a compressed tar file. This is called by <i>vic-switchboard</i> , in a response to a Bluetooth LE log command.

Table 330: Vector diagnostic & logging software

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.

<i>displayFaultCode</i>	Displays error fault codes on the LCD. This is not called; see <i>vic-faultCodeDisplay</i> .
<i>fault-code-clear</i>	This clears and pending or displayed faults (by deleting the relevant files). This allows new fault code to be displayed. This is called by <i>vic-init.sh</i> .
<i>fault-code-handler</i>	XXX This is called by the fault-code service.
<i>librobotLogUploader.so</i>	Sends logs to cloud. This library is employed by <i>libcozmo_engine</i> , <i>vic-gateway</i> and <i>vic-log-upload</i> .
<i>libosState</i>	Used to profile the CPU temperature, frequency, load; the WiFi statistics, and ETC. This is used by <i>libvictor_web_library</i> , <i>vic-anim</i> , and <i>vic-dasmgr</i> .
<i>libwhiskeyToF</i>	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 <i>libcozmo_engine</i> .
<i>rampost</i>	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.
<i>vic-anim</i>	Includes the support for the Customer Care Information Screen. This is started by the <i>vic-anim</i> service.
<i>vic-crashuploader-init</i>	Removes empty crash files, renames the files ending in “.dmp~” to “.dmp”. This is called by the <i>vic-crashuploader</i> service.
<i>vic-crashuploader</i>	A script that sends crash mini-dump files to backtrace.io. This is called by the <i>vic-crashuploader</i> service.
<i>vic-dasmgr</i>	This is started by the <i>vic-dasmgr</i> service.
<i>vic-faultCodeDisplay</i>	Displays error fault codes on the LCD. This is called by <i>fault-code-handler</i> .
<i>vic-init.sh</i>	Takes the log messages from <i>rampost</i> and places them into the system log, forwards any kernel panics. This is started by the <i>vic-init</i> service.
<i>vic-log-event</i>	A program that is passed an event code in the command line. This is called by TBD.
<i>vic-log-forward</i>	This is called by <i>vic-init.sh</i>
<i>vic-log-kernel-panic</i>	This is called by <i>vic-init.sh</i>
<i>vic-log-upload</i>	This is called by <i>vic-log-uploader</i>
<i>vic-log-uploader</i>	“This script runs as a background to periodically check for outgoing files and attempt to upload them by calling ‘ <i>vic-log-upload</i> ’.” This is started by the <i>vic-log-uploader</i> service.
<i>vic-logmgr-upload</i>	“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.
<i>vic-on-exit</i>	Called by <i>systemd</i> after any service stops. This script places the fault code associated with the service (if another fault code is not pending) into <i>/run/fault_code</i> for display.
<i>vic-powerstatus.sh</i>	Record every 10 seconds the CPU frequency, temperature and the CPU & memory usage of the “ <i>vic-</i> ” 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).

119. 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

119.1. 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

119.2. 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.

119.3. 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.

119.4. 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 bugs⁵¹.

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.

119.5. “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.

120. 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.

121. 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 14, *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 14, *Upload Debug Logs*.

122. 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

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.”

122.1. 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:

File	Description
<i>connman-services.txt</i>	connmanctl services
<i>dmesg.txt</i>	dmesg
<i>ifconfig.txt</i>	ifconfig wlan0
<i>iwconfig.txt</i>	iwconfig wlan0
<i>log.txt</i>	Concatenates /var/log/messages.1.gz (uncompressed) and /var/log/messages
<i>netstat-ptlnu.txt</i>	netstat -ptlnu
<i>ping-anki.txt</i>	Ping's anki.com for connectivity and latency.
<i>ping-gateway.txt</i>	Looks up the IP address (using netstat) of the gateway that Vector is using and pings it for connectivity and latency.
<i>ps.txt</i>	Process stats (ps) of Anki's "Vic" processes
<i>top.txt</i>	top -n 1

Table 331: Files in the log archive

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

123. 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
<i>channels</i>	array	An array of the channel logging enable structures
<i>levels</i>	array	An array of logging level enable structures

Table 332: The console filter channel structure

⁵² The other OS key is "osx" which suggests that Vector's software was development on an OS X platform.

This “channels” is as an array of structures with the following fields:

Field	Type	Description & Notes
<i>channel</i>	string	The name of the channel
<i>enabled</i>	boolean	True if should log information from the channel, false if not.

Table 333: The channel logging enable structure

This “levels” is an array of structures with the following fields:

Field	Type	Description & Notes
<i>enabled</i>	boolean	True if should log information at that level, false if not.
<i>level</i>	string	“event” or “debug”

Table 334: The logging level enable structure

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.

Channel	enabled	Description & Notes
<i>Actions</i>	false	
<i>AIWhiteboard</i>	false	
<i>Alexa</i>	false	
<i>Audio</i>	false	
<i>Behaviors</i>	false	
<i>BlockPool</i>	false	
<i>BlockWorld</i>	false	
<i>CpuProfiler</i>	true	
<i>FaceRecognizer</i>	false	
<i>FaceWorld</i>	false	
<i>JdocsManager</i>	true	
<i>MessageProfiler</i>	true	
<i>Microphones</i>	false	
<i>NeuralNets</i>	false	
<i>PerfMetric</i>	true	
<i>SpeechRecognizer</i>	false	
<i>VisionComponent</i>	false	
<i>VisionSystem</i>	false	
*	false	

Table 335: The channels

124. 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.

124.1. 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.

⁵³ This is a very helpful feature

124.2. 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.
- 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.

Speculated purpose: To identify typical and on unusual processing loads and temperatures. The heavy uses cases are likely undesired and would be something to identify.

The data gather in Vector for these is primarily based in a library called libosState.

124.2.1 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:

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

Table 336: The WiFi related stats /proc files

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

124.2.2 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:

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	

Table 337: Named device and control files

How this is used: This information was probably intended to find the balance between overheating and AI performance.

124.3. 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:

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.

Table 338: The experiments TBD structure

The meta structure has the following fields:

Field	Type	Description & Notes
project_id	string	“cozmo” ⁵⁴
revision	int	1
version	int	2

Table 339: The meta JSON structure

⁵⁴ I suspect that this would have changed once experiments were initiated with Vector

An experiment structure has the following fields:

Field	Type	Description & Notes	Table 340: The experiment JSON structure
<i>activation_mode</i>	string	“automatic”	
<i>audience_tags</i>	array of TBD		
<i>forced_variations</i>	array of TBD		
<i>key</i>	string	“report_test_auto”	
<i>pop_frac_pct</i>	int	Portion of the population, as a percentage, that will take part in this experiment.	
<i>pause_time_utc_iso8601</i>	string		
<i>resume_time_utc_iso8601</i>	string		
<i>start_time_utc_iso8601</i>	string	The date and time that the experiment will commence.	
<i>stop_time_utc_iso8601</i>	string	The date and time that the experiment will end.	
<i>variations</i>	array of variation		
<i>version</i>	int	0	

A variation structure has the following fields:

Field	Type	Description & Notes	Table 341: The variation JSON structure
<i>key</i>	string	One of at least two populations subject to the test: “control” or “treatment”	
<i>pop fract pct</i>	int	Portion of the population, as a percentage, that will be in this subject group.	

125. 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.

126. CREDITS

Credit and thanks to Anki, CORE, Melanie T for access to the flash partitions, file-systems, decode keys, board shots, and information on the motor assembly. Fictiv for additional board shots. The board shots helped identify parts on the board and inter-connection on the board. Alexander Entinger for connector signal information. HSReina for Bluetooth LE protocol information. Wayne Venables for crafting a C# version of the SDK. Some drawings adapted from Steph Dere, and Jesse Easley's twitter.

127. REFERENCE DOCUMENTATION AND RESOURCES

127.1. 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>

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

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

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

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>

127.2. OTHER

cozmopedia.org

Entinger, Alexander; *Anki Vector base-board connector*
<https://github.com/aentinger/anki-vector-baseboard>

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/>

Tenchov, Kaloyan; *PyCozmo*
<https://github.com/zayfod/pycozmo/tree/master/pycozmo>

Venable, Wayne; *Anki.Vector.SDK*
<https://github.com/codaris/Anki.Vector.SDK>
<https://github.com/codaris/Anki.Vector.Samples>
<https://weekendrobot.com/>

Zaks, Mazim *FlatBuffers Explained*, 2016-Jan-30
<https://github.com/mzaks/FlatBuffersSwift/wiki/FlatBuffers-Explained>

127.3. 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>

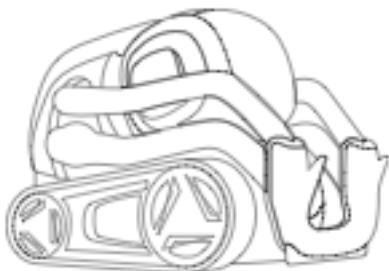
Qualcomm, *DragonBoard™ 410c based on Qualcomm® Snapdragon™ 410E processor ADB Debugging Commands Guide*, LM80-P0436-11, Rev C, 2016 Sep
lm80-p0436-11_adb_commands.pdf

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	<i>unknown (diagnostic/data analytics service?)</i>
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 342: 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)

Phrase	Description	Table 343: Glossary of common terms and phrases
<i>A*</i>	A path finding algorithm	
<i>aboot</i>	The Android boot-loader used to launch Vector's linux system.	
<i>accelerometer</i>	A sensor used to measure the angle of Vector's head, and acceleration (change in velocity).	
<i>animation</i>	"a sequence of highly coordinated movements, faces, lights, and sounds ... to demonstrate an emotion or reaction."	
<i>attitude</i>	Vector's orientation, esp relative to the direction of travel	
<i>beam forming</i>	A technique using multiple microphones to listen to a single speaker by selectively paying attention to sound only coming from that direction.	
<i>behavior</i>	"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 <code>drive_on_charger</code> , <code>set_lift_height</code> , etc."	Anki SDK
<i>boot loader</i>	A piece of software used to load and launch the application software.	
<i>C-like abstract data structure (CLAD)</i>	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 developed.)	
<i>capacitive touch</i>	A type of sensing where light contact, such as touch, is detected without requiring pressing a mechanism.	
<i>cascade</i>	Applies a series of fast to compute filters and classifiers to detect low-level features and identify things like faces.	
<i>certificate</i>	Vector generates an SSL certificate that can be used for the secure communications.	
<i>characteristic (Bluetooth LE)</i>	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>

<i>client token</i>	A string token provided by Vector that is passed with each SDK command.
<i>control</i>	Responsible for motors and forces to move where and how it is told to. (smooth arcs)
<i>D*-lite</i>	A path-finding algorithm
<i>device mapper verity (dm-verity)</i>	A feature of the Linux kernel that checks the boot and RAM file systems for alteration, using signed keys
<i>entitlement</i>	An entitlement is a family of features or resources that the program or owner is allowed to use.
<i>face detection</i>	The ability to realize that there is a face in the image, and where it is
<i>face recognition</i>	The ability to know the identity of a face seen.
<i>feature flags aka feature toggle</i>	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.
<i>field of view</i>	How wide of an area in the world that the camera can see
<i>firmware</i>	A type of software held (and usually executed from) in ROM or flash. It may have a (minimal) operating system, but often does not.
<i>flash</i>	A type of persistent (non-volatile) storage media.
<i>guidance</i>	Builds the desired path
<i>gyroscope</i>	A sensor that is used to measure how fast Vector is turning (the angular velocity) along its x, y, and z axes.
<i>Haar feature</i>	Facial features picked out using Haar wavelets
<i>Haar wavelet</i>	A fast, low-cost that can be used to pick out (or recognize) simple features in an image.
<i>inner node</i>	A node in a tree data structure that does links to other nodes below it. Often it does not hold any other information.
<i>intent</i>	This is an internal code used to represent the command or question that corresponds to a phrase spoken by a person.
<i>Kalman filter</i>	Used to merge two or more noisy signals together to estimate a proper signal.
<i>leaf node</i>	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.
<i>navigation</i>	Knowing where it is in the map
<i>nonce</i>	An initially random number, incremented after each use.
<i>path planning</i>	Forms smooth arcs and line segments to move in around an environment to avoid collisions, blocked paths, and cliffs.
<i>pose</i>	The position and orientation of an object relative to a coordinate system
<i>power source</i>	Where the electric energy used to power Vector comes from.
<i>quad-tree</i>	A way of compressing a 2D map down into regions.
<i>rapidly-expanding random tree</i>	A path-finding algorithm
<i>recovery mode</i>	A separate, independent operating system that Vector can boot into for purposes of downloading software to replace a damaged partition.

<i>robot name</i>	Vector's robot name looks like "Vector-E5S6". It is "Vector-" followed by a 4 letters and numbers.
<i>session token</i>	A string token provided by the Anki servers that is passed to Vector to authenticate with him and create a <i>client token</i> .
<i>simultaneous localization and mapping</i>	A vision-based technique for building a map of the immediate world for purposes of positioning oneself within it and detecting relative movements.
<i>service (Bluetooth LE)</i>	A key-value table grouped together for a common purpose. A service is uniquely identified by its UUID.
<i>software</i>	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.
<i>syscon</i>	The name of the firmware program running on the base-board.
<i>text to speech</i>	A process of reading aloud a word, phrase, sentence, etc.
<i>trigger word</i>	aka wake word
<i>Trust Zone</i>	A security mode on ARM processor where privileged/special code is run. This includes access to encryption/decryption keys.
<i>universally unique identifier (UUID)</i>	A 128bit number that is unique. (effectively same as GUID)
<i>wake word</i>	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.

Tool	Description
<i>Acapela</i>	Vector uses Acapela's text to speech synthesizer, and the Ben voice. https://www.acapela-group.com/
<i>Advanced Linux Sound Architecture (alsa)</i>	The audio system https://www.alsa-project.org
<i>Amazon Alexa</i>	A set of software tools that allows Vector to integrate Alexa voice commands, probably in the AMAZONLITE distribution https://github.com/anki/avs-device-sdk https://developer.amazon.com/alexa-voice-service/sdk
<i>Amazon Simple Queue Service (SQS)</i>	Vector employs Amazon's SQS for its DAS functions.
<i>Amazon Simple Storage Service (S3)</i>	Vector's cloud interface uses Amazon's AWS go module to interact with Amazon's service: 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
<i>Amazon Web services</i>	used on the server https://aws.amazon.com/
<i>android boot-loader</i>	Vector uses the Android Boot-loader; the code can be found in the earlier archive.
<i>ARM NN</i>	ARM's neural network support https://github.com/ARM-software/armnn
<i>AudioKinetic Wwise</i> ⁵⁶	Used to craft the sounds https://wwwaudiokinetic.com/products/wwise/
<i>Backtrace.io</i>	A service that receives uploaded minidumps from applications in the field and provides tools to analyze them. https://backtrace.io
<i>clang</i>	A C/C++ compiler, part of the LLVM family https://clang.llvm.org
<i>bluez v5</i>	Bluetooth LE support http://www.bluez.org/
<i>busybox</i>	The shell on the Anki Vector linux https://busybox.net
<i>chromium update</i>	?

Table 344: Tools used by Anki

⁵⁶ <https://blog.audiokinetic.com/interactive-audio-brings-cozmo-to-life/>

<i>civetweb</i>	The embedded webserver that allows Mobile apps and the python SDK to communicate with Vector. https://github.com/civetweb/civetweb
<i>connman</i>	Connection manager for WiFi https://01.org/connman
<i>GNU C Compiler (gcc)</i>	GCC version 4.9.3 was used to compile the kernel
<i>golang</i>	Go is used on the server applications, and (reported) some of Vector's internal software.
<i>Google Breakpad</i>	Google Breakpad is used to generate tracebacks and mini-dump files of programs that crash. Results are sent to http://backtrace.io https://chromium.googlesource.com/breakpad/breakpad
<i>Google FlatBuffers</i>	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" ⁵⁷ https://github.com/google/flatbuffers
<i>Google Protobuf</i>	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. https://developers.google.com/protocol-buffers
<i>Google RPC (gRPC)</i>	A "remote procedure call" standard, that allows mobile apps and the python SDK to communicate with Vector. https://grpc.io/docs/quickstart/cpp/
<i>hdr-histogram</i>	Unknown use https://github.com/HdrHistogram/HdrHistogram
<i>libsodium</i>	Cryptography library suitable for the small packet size in Bluetooth LE connections. Used to encrypt the mobile applications Bluetooth LE connection with Vector. https://github.com/jedisct1/libsodium
<i>linux, yocto</i> ⁵⁸	The family of linux distribution used for the Anki Vector (v3.18.66)
<i>linux</i>	on the server
<i>linux unified key storage (LUKS)</i>	
<i>Maya</i>	A character animation tool set, used to design the look and movements of Cozmo and Vector. The tool emitted the animation scripts.
<i>mpg123</i>	A MPEG audio decoder and player. This is needed by Alexa; other uses are unknown. https://www.mpg123.de/index.shtml
<i>ogg vorbis</i>	Audio codec https://xiph.org/vorbis
<i>Omron OKAO Vision</i>	Vector uses the Omron Okao Vision library for face recognition and tracking. https://plus-sensing.omron.com/technology/position/index.html
<i>open CV</i>	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/
<i>openssl</i>	used to validate the software update signature https://www.openssl.org
<i>opkg</i>	Package manager, from yocto https://git.yoctoproject.org/cgit.cgi/opkg/
<i>Opus codec</i>	Audio codec; to encode speech sent to servers http://opus-codec.org/
<i>perl</i>	A programming language, on Victor https://www.perl.org
<i>Pretty Fast FFT</i> <i>pfft</i>	Julien Pommier's FFT implementation for single precision, 1D signals https://bitbucket.org/jpommier/pfft
<i>Pryon, Inc</i>	The recognition for the Alexa keyword at least the file system includes the same model as distributed in AMAZONLITE https://www.pryon.com/company/
<i>python</i>	A programming language and framework used with desktop tools to communicate with Vector. Vector has python installed. Probably used on the server as well. https://www.python.org
<i>Qualcomm</i>	Qualcomm's device drivers, camera support and other kit are used.
<i>Segger ICD</i>	A high-end ARM compatible in-circuit debugging probe. Rumoured to have been used by Anki engineers, probably with the STM32F030 https://www.segger.com/products/debug-probes/j-link/
<i>Sensory TrulyHandsFree</i>	Vectors recognition for "Hey Vector" and Alexa wake word is done by Sensory, Inc's TrulyHandsfree SDK 4.4.23 (c 2008) https://www.sensory.com/products/technologies/trulyhandsfree/ https://en.wikipedia.org/wiki/Sensory,_Inc.
<i>Signal Essence</i>	Designed the microphone array, and the low-level signal processing of audio input. https://signalessence.com/
<i>Sound Hound, inc</i>	Vector's Q&A "knowledge graph" is done by Sound Hound https://blog.soundhound.com/hey-vector-i-have-a-question-3c174ef226fb
<i>SQLite</i>	This is needed by Alexa; other uses are unknown https://www.sqlite.org/index.html
<i>systemd</i>	Used by Vector to launch the internal services https://www.freedesktop.org/software/systemd/
<i>tensor flow lite (TFLite)</i>	TensorFlow lite is used to recognize hands, the desk surface, and was intended to support recognizing pets and common objects. https://www.tensorflow.org/lite/microcontrollers/get_started

128. 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:

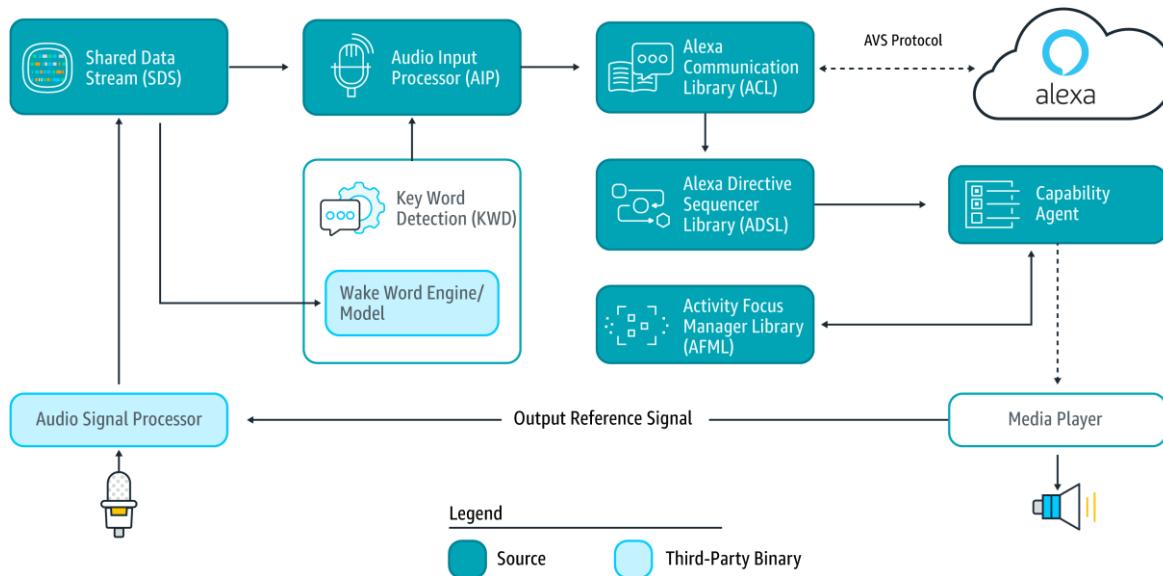


Figure 69: Alexa's function blocks
(image courtesy Amazon)

Alexa's modules include:

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

Table 345: Alexa files

	and controlling audio playback.”
<code>libAudioResources.so</code>	Alexa’s audio resources, including calls
<code>libAVSCommon.so</code>	Alexa’s voice service support
<code>libAVSSystem.so</code>	Alexa’s voice service support
<code>libCapabilitiesDelegate.so</code>	Alexa capabilities. “Handles Alexa-driven interactions; specifically, directives and events. Each capability agent corresponds to a specific interface exposed by the AVS API.”
<code>libCBLAuthDelegate.so</code>	Alexa Authorization
<code>libCertifiedSender.so</code>	Alexa certified sender
<code>libContextManager.so</code>	Alexa’s context manager
<code>libESP.so</code>	Alexa ESP, Dummy ESP
<code>libInteractionModel.so</code>	“This interface allows a client to support complex interactions initiated by Alexa, such as Alexa Routines.”
<code>libNotifications.so</code>	Alexa Notifications. “The interface for displaying notifications indicators.” Uses SQLite
<code>libPlaybackController.so</code>	“The interface for navigating a playback queue via GUI or buttons.”
<code>libPlaylistParser.so</code>	Alexa playlist
<code>libRegistrationManager.so</code>	Alexa’s registration manager
<code>libSettings.so</code>	Alexa’s settings & preferences module
<code>libSpeakerManager.so</code>	
<code>libSpeechSynthesizer.so</code>	“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:

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.”

Table 346: The system fault codes

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

Status	Meaning
200	The TAR contents did not follow the expected order.
201	Unhandled section format for expansion, or The manifest version is not supported, or The OTA has the wrong number of images for the type, or The OTA is missing a BOOT or SYSTEM image, or The manifest configuration is not understood
202	Could not mark target, a, or b slot unbootable, or Could not set target slot as active
203	Unable to construct automatic update URL, or The URL could not be opened

Table 347: OTA update-engine status codes

204	The file wasn't a valid TAR file, or is corrupt
205	The compression scheme is not supported, or Decompression failed, the file may be corrupt
207	Delta payload error
208	Couldn't sync OS images to disk, or Disk error while transferring OTA file.
209	The manifest failed signature validation; or the <code>aboot</code> , <code>boot image</code> , <code>system image</code> , or <code>delta.bin</code> 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 partitions⁶⁰ and file systems. Many are probably not used. Quotes are from Android documentation.

The file system table tells us where they are stored in the partitions, and if they are non-volatile.

Mount point	Partition name	Description & Notes	<i>Table 348: The file system mount table</i>
/	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 /run /var/volatile /dev/sm		Internal temporary file systems; holds temporary files, interprocess communication	

The partition table⁶² found on the Vector:

Partition name	Size	Description & Notes	<i>Table 349: The partition table</i>
ABOOT	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.	
ABOOTBAK	1 MB		
BOOT_A	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.	
BOOT_B	32 MB		
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. In typical aboot implementations this partition is used to hold "device information including: is_unlocked (aboot), is_tampered, is_verified,	

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>

		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	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.
MODEMST2	1.5MB	
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	512KB	The primary and backup partitions for resource and power management. This is in the format of a signed, statically linked ELF binary.
RPMBAK	512KB	
SBL1	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.
SBL1BAK	512KB	
SEC	16KB	The secure boot fuse settings, OEM settings, signed-bootloader stuff
SSD	8KB	“Secure software download” for secure storage, encrypted RSA keys, etc
SYSTEM_A	896MB	The primary and backup system applications and libraries with application specific code. Updates modify the non-active partition, and then swap which one is active.
SYSTEM_B	896MB	
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:

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/connman	
/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

Table 350: Files

<code>/dev/socket/_engine_gateway_proto_server_</code>	The IPC socket interface to Vector's Gateway [TBD] server
<code>/dev/socket/_engine_switch_server_</code>	The IPC socket interface to Vector's Switchbox [TBD] server
<code>/etc/os-version</code>	Contains the OS (linux) version string.
<code>/etc/os-version-rev</code>	Contains the OS (linux) revision string.
<code>/factory/cloud/something.pem</code>	
<code>/proc/sys/kernel/random/boot_id</code>	A random identifier, created each boot
<code>/sys/devices/system/cpu/possible⁶³</code>	The number of CPUs and whether they can be used.
<code>/sys/devices/system/cpu/present</code>	
<code>/run/after_maintenance_reboot</code>	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 <code>/data/maintenance_reboot</code> had been set.
<code>/run/anki-crash-log</code>	
<code>/run/das_allow_upload</code>	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.
<code>/run/fake-hwclock-cmd⁶⁴</code>	Sets the fake time to the time file (Vector doesn't have a clock)
<code>/run/fault_code</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.
<code>/run/fault_code.pending</code>	The next fault code in queue to be handled
<code>/run/fault_code.showing</code>	The fault code being displayed
<code>/run/fault_restart_count</code>	This is incremented with each restart, and cleared by a reboot.
<code>/run/fault_restart_uptime</code>	
<code>/tmp/data_cleared</code>	
<code>/tmp/vision/neural_nets</code>	

Key named device files employed in Vector binaries:

Table 351: Named device and control files

File	Description
<code>/dev/fb0</code>	The display framebuffer
<code>/dev/spidev0.0</code>	The SPI channel to communicate with the IMU
<code>/dev/spidev1.0</code>	The SPI channel to communicate with the LCD
<code>/dev/ttyHS0</code>	Serial connection with the base-board
<code>/dev/ttyHSL0</code>	Console log
<code>/sys/class/android_usb/android0/iSerial</code>	Set to Vector's serial number
<code>/sys/class/gpio/gpio83</code>	Used to control the camera power
<code>/sys/class/leds/face-backlight-left/brightness</code>	LCD left backlight control
<code>/sys/class/leds/face-backlight-right/brightness</code>	LCD right backlight control
<code>/sys/devices/platform/soc/1000000.pinctrl/gpio/gpiochip0/base</code>	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>

<code>/sys/devices/system/cpu/cpu0/cpufreq/scaling_max_freq</code>	The maximum frequency that the CPU can run at. Initially set to 533MHz
<code>/sys/kernel/debug/msm_otg/bus_voting</code>	Disabled to prevent the USB from pinning RAM to 400MHz.
<code>/sys/kernel/debug/rpm_send_msg/message</code>	Used to control the RAM controller. The RAM is set to a maximum of 400MHz.
<code>/sys/devices/soc/1000000.pinctrl/gpio/gpiochip0/base</code>	LCD backlight enable (left or right?) GPIO config
<code>/sys/devices/soc.0/1000000.pinctrl/gpio/gpiochip911/base</code>	LCD backlight enable (left or right?) GPIO config
<code>/sys/module/spidev/parameters/bufsiz</code>	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.

129. CUBE SERVICES

times and other feature parameters:

Service	UUID ⁶⁵	Description & Notes
<i>Device Info Service</i> ⁶⁶	180A ₁₆	Provides device and unit specific info –it's manufacturer, model number, hardware and firmware versions
<i>Generic Access Profile</i> ⁶⁷	1800 ₁₆	The device name, and preferred connection parameters
<i>Generic Attribute Transport</i> ⁶⁸	1801 ₁₆	Provides access to the services.
<i>Cube's Service</i>	C6F6C70F-D219-598B-FB4C-308E1F22F830 ₁₆	Service custom to the cube, reporting battery, accelerometer and date of manufacture

Table 352: 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
<i>Device Name (Default)</i>	"Vector Cube"
<i>Firmware Revision</i>	"v_5.0.4"
<i>Manufacturer Name</i>	"Anki"
<i>Model Number</i>	"Production"
<i>Software Revision</i>	"2.0.0"

Table 353: 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

129.1. CUBE'S ACCELEROMETER SERVICE

Values are little-endian, except where otherwise stated.

UUID	Access	Size	Notes	
0EA75290-6759-A58D-7948-598C4E02D94A ₁₆	Write	unknown		Table 354: Cube's accelerometer service characteristics
450AA175-8D85-16A6-9148-D50E2EB7B79E ₁₆	Read		The date and time of manufacture (?) char[] <i>A date and time string</i>	
43EF14AF-5FB1-7B81-3647-2A9477824CAB ₁₆	Read, Notify, Indicate		Reads the battery and accelerometer uint16_t <i>battery ADC value</i> uint16_t <i>accelerometer X ADC value #1</i> uint16_t <i>accelerometer Y ADC value #1</i> uint16_t <i>accelerometer Z ADC value #1</i> uint16_t <i>accelerometer X ADC value #2</i> uint16_t <i>accelerometer Y ADC value #2</i> uint16_t <i>accelerometer Z ADC value #2</i> uint16_t <i>accelerometer X ADC value #3</i> uint16_t <i>accelerometer Y ADC value #3</i> uint16_t <i>accelerometer Z ADC value #3</i>	
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

130. VECTOR SERVICES SERVICE

times and other feature parameters:

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.

Element	Value	
Device Name (Default)	“Vector” followed by his serial number	Table 356: The Vector's Device info settings

⁶⁹ All values are a little endian, per the Bluetooth 4.0 GATT specification

130.1. 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 357: Vector's serial service characteristics

APPENDIX G

Servers & Data Schema

This Appendix describes the servers that Vector contacts⁷⁰

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:

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:

Server	Description & Notes
https://api.amazon.com/auth/O2/	Used to authenticate the account for the Alexa

Table 358: The servers that Vector contacts.

Table 359: The servers that the mobile application contacts.

Table 360: The Amazon Alexa Voice Service servers that Vector contacts.

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:

Feature	enabled	Description & Notes
<i>ActiveIntentFeedback</i>	true	
<i>Alexa</i>	true	The ability to use Alexa
<i>Alexa_AU</i>	true	The ability to use Alexa, localized for Australia
<i>Alexa_UK</i>	true	The ability to use Alexa, localized for the UK
<i>AttentionTransfer</i>	false	
<i>CubeSpinner</i>	false	
<i>Dancing</i>	true	The ability for Vector to dance to music.
<i>Exploring</i>	true	The ability for Vector to explore his area
<i>EyeColorVC</i>	true	The ability to set Vector's eye color through a voice command
<i>FetchCube</i>	true	The ability for Vector to fetch his cube
<i>FindCube</i>	true	The ability for Vector to find his cube
<i>GazeDirection</i>	false	
<i>GreetAfterLongTime</i>	true	
<i>HandDetection</i>	true	The ability for Vector to spot hands
<i>HeldInPalm</i>	true	
<i>HowOldAreYou</i>	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?"
<i>Invalid</i>	false	
<i>Keepaway</i>	true	
<i>KnowledgeGraph</i>	true	The ability for Vector to answer a question when asked "Hey Vector, I have a question..."
<i>Laser</i>	false	
<i>Messaging</i>	false	
<i>MoveCube</i>	true	
<i>PopAWheelie</i>	true	The ability for Vector pop a wheelie using his cube
<i>PRDemo</i>	false	

Table 361: The features

<i>ReactToHeldCube</i>	true	
<i>ReactToIllumination</i>	true	
<i>RollCube</i>	true	The ability for Vector to drive up and roll his cube
<i>StayOnChargerUntilCharged</i>	true	
<i>TestFeature</i>	false	
<i>Volume</i>	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:

Intent	Enumeration	Phrase	Table 362: The "Hey Vector" phrases
<i>movement_backward</i>	23	Back up	
<i>imperative_scold</i>	18	Bad robot	
<i>imperative_quiet</i>		Be quiet	
<i>global_stop</i>	3	Cancel the timer	
		Change/set your eye color to [blue, green, lime, orange, purple, sapphire, teal, yellow].	
<i>check_timer</i>	1	Check the timer	
<i>imperative_come</i>	10	Come here	
<i>imperative_dance</i>	11	Dance.	
<i>play_popawheelie</i>	34	Do a wheelstand	
<i>imperative_fetchcube</i>	12	Fetch your cube	
<i>imperative_findcube</i>	13	Find your cube	
<i>play_fistbump</i>	32	Fist Bump	
<i>play_fistbump</i>	32	Give me a Fist Bump	
<i>movement_backward</i>	23	Go backward	
<i>explore_start</i>	2	Go explore	
<i>movement_forward</i>	22	Go forward.	
<i>movement_turnleft</i>	24	Go left	
<i>movement_turnright</i>	25	Go right	
<i>system_sleep</i>		Go to sleep	
<i>system_charger</i>		Go to your charger	
		Good afternoon	
<i>greeting_goodbye</i>	4	Goodbye	

	Good evening
<i>greeting_goodnight</i>	Good night
<i>greeting_goodmorning</i>	5 Good morning
<i>imperative_praise</i>	16 Good robot
<i>seasonal_happyholidays</i>	36 Happy Holidays
<i>seasonal_happynewyear</i>	37 Happy New Year
<i>greeting_hello</i>	6 Hello
	He's behind you
<i>character_age</i>	0 How old are you
<i>imperative_abuse</i>	7 I hate you.
<i>knowledge_question</i>	27 I have a question ...
<i>imperative_love</i>	15 I love you.
<i>imperative_apology</i>	9 I'm sorry.
<i>play_blackjack</i>	31 Let's play Blackjack
	Listen to music
<i>imperative_lookatme</i>	14 Look at me
	Look behind you
	My name is [Your Name]
<i>imperative_negative</i>	17 No
<i>play_pickupcube</i>	33 Pick up your cube.
<i>play_anygame</i>	29 Play a game
<i>play_anytrick</i>	30 Play a trick
<i>play_blackjack</i>	31 Play Blackjack
<i>play_popawheelie</i>	34 Pop a wheelie.
<i>play_rollcube</i>	35 Roll your Cube
<i>imperative_quiet</i>	Quiet down
	Run
<i>set_timer</i>	38 Set a timer for [length of time]
<i>imperative_shutup</i>	Shut up
<i>explore_start</i>	2 Start Exploring
	Stop Exploring
<i>global_stop</i>	3 Stop the timer
<i>take_a_photo</i>	40 Take a picture of [me/us]
<i>take_a_photo</i>	40 Take a picture
<i>take_a_photo</i>	40 Take a selfie

<i>movement_turnaround</i>	26	Turn around
<i>movement_turnleft</i>	24	Turn left
{same as be quiet }		Turn off
<i>movement_turnright</i>	25	Turn right
<i>imperative_volumelvel</i>	19	Volume [number].
<i>imperative_volumedown</i>	21	Volume down
<i>imperative_volum eup</i>	20	Volume up.
		Volume maximum
<i>names_ask</i>	28	What's my name?
<i>weather_response</i>	41	What's the weather in [City Name]?
<i>weather_response</i>	41	What's the weather report?
<i>show_clock</i>	39	What time is it?
<i>blackjack_hit</i>		
<i>blackjack_playagain</i>		
<i>blackjack_stand</i>		
<i>global_delete</i>		
<i>imperative_lookoverthere</i>		
<i>knowledge_response</i>		
<i>knowledge_unknown</i>		
<i>meet_victor</i>		
<i>message_playback</i>		
<i>message_record</i>		
<i>silence</i>		
<i>status_feeling</i>		
<i>imperative_affirmative</i>	8	Yes

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

Questions

Subject	Example Phrase
<i>Current conversion</i>	What's 1000 Yen in US Dollars?
<i>Flight status</i>	What is the status of American Airlines Flight 100?
<i>Equation solver</i>	What is the square root of 144?
<i>General knowledge</i>	What is the tallest building?
<i>places</i>	What is the distance between London and New York?
<i>People</i>	Who is Jarvis?

Table 363: The Vector questions phrases

<i>Nutrition</i>	How many calories are in an avocado?
<i>Sports</i>	Who won the World Series?
<i>Stock market</i>	How is the stock market?
<i>Time zone</i>	What time is it in Hong Kong?
<i>Unit conversion</i>	How fast is a knot?
<i>Word definition</i>	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 diagnostics / analytics services (see Chapter 20)

131. DAS TRACKED EVENTS AND STATISTICS

131.1. BASIC INFORMATION

131.1.1 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 364: Version info, posted to DAS

131.1.2 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 365: Start up information, posted to DAS

131.1.3 Start-up Information not described elsewhere

The following are start-up events that are posted to the diagnostic logger:

Event	Description & Notes
ntp.timesync	
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	

Table 366: Settings and preferences, posted to DAS

Note: other startup events are covered elsewhere with their functional groups.

131.2. 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 367: Power management events, posted to DAS

131.2.1 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 368: Battery level events and statistics

131.2.2 Charger Statistics and Events

The charging function of the battery management system posts the following events and state information:

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	

Table 369: Charger statistics and events, posted to DAS

131.3. MOTOR AND IMU STATISTICS AND EVENTS

The motor controllers post the following events and statistics:

Event	Description & Notes
calibrate_motors	
head_motor_calibrated	
head_motor_uncalibrated	
imu_filter.gyro_calibrated	
lift_motor_calibrated	
lift_motor_uncalibrated	

Table 370: Motor events

131.4. COMMUNICATION RELATED EVENTS POSTED TO DAS

131.4.1 Base-Board / Spine Related Events Posted to DAS

The communication with the base-board controller posts the following 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.

Table 371: Base-board / spine related DAS events

rampost.spine.select_timeout	There was a timeout in communicating with rampost the base-board.
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

131.4.2 Accessory-Related Events Posted to DAS

The communication with the mobile application and SDK posts the following 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	

Table 372: Accessory cube related DAS events

Note: see the updates section for events related to updating the cube firmware

131.4.3 Mobile-App / SDK Related Events Posted to DAS

The communication with the mobile application and SDK posts the following events:

Event	Description & Notes
ble.connection	
ble_conn_id.stop	
ble.disconnection	
wifi.initial_state	

Table 373: Mobile application / SDK related DAS events

131.5. UPDATE-RELATED EVENTS POSTED TO DAS

The following are events are posted by the update subsystem:

Event	Description & Notes
cube.firmware_flash_success	
rampost.dfu.desired_version	

Table 374: Update events

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	

131.6. BEHAVIOUR, FEATURE, MOOD, AND ENGINE RELATED EVENTS POSTED TO DAS

The engine/animation controller post the following behavior-related 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	

Table 375: Behaviour, feature, mood and engine related DAS events

131.7. ROBOT LIFETIME STATISTICS & EVENTS

Vector tracks the following behavior and events

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	

Table 376: The robot lifetime stats schema

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

Annunciators 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

131.8. 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 sales⁷⁴

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

131.9. 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-farewell-lecture-by-john-sosoka-former-cto-of-ugobeJohnSosoka>