QCAN2

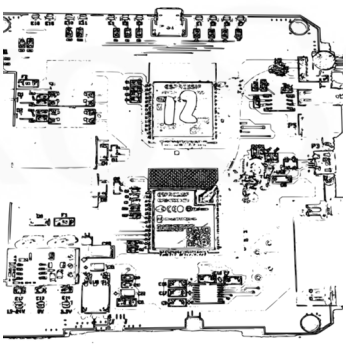
Document under process of revision, once released, this notice will be removed.

Akostar Developer's Manual

Document Version 3.6

Intellectual property notice:

All ideas, concepts, designs, and arrangements communicated within this document are preliminary, and they are subject to change. Information contained within may be proprietary, only to be shared or used upon the approval of respective owners / holders.



Revision and updates:

The developer and manufacturer reserves the right to change and review the design and implementation of said device, at the sole discretion of the intellectual property owner. In case of dispute, the order of resolution is fitness for purpose, compatibility, functionality, aesthetics.

Document Revision History:

Doc Revision	Date	By	Description	Notes
1.0	Mar-13-2019	Peter Glen	Initial Write-up	
1.2	Oct-2019 / Dec-2020	Team	Code base	
1.x	Dec-2019 / Jan-2020	Team	Updated descriptions	
2.0	Jan-15-2020	Peter Glen / Chris Bogue	Review	
3.0	Mar-24-2020	Peter Glen	Doc release	
3.1	Mar-31-2020	Peter Glen	Master Doc	Re-structure
3.2	Apr-22-2020	Peter Glen	Doc for Rel #5	Firmware 1.10
3.3	May-5-2020	Peter Glen	Doc for Rel #6	Firmware 1.11
3.4	May-20-2020	Peter Glen / Chris Bogue	Doc for Rel #8,9	Firmware 1.14
3.5	Jul-1-2020	Added Rem Call	Doc for Rel #10	Firmware 1.18
3.6	Dec-18-2020	Finalized some	Doc for Rel #14	Firmware 2.0
3.6	Mar-15-2021	Release to Savant	Doc for Rel #15	Firmware 2.2

Table of Contents

This part of the page is left intentionally blank

QCAN2 functional introduction

*This is a collection of notes produced during implementation. Comments are welcome. (Items that are marked 'implemented' are left there for informational purposes.) These implementation notes may also provide useful information on troubleshooting, and give an insight to the internal workings of the QCAN2. The document also contains notes that where exchanged between Savant and Akostar. If there is a conflict within the in contents, it is resolved by giving items marked '**update**' priority.*

The previous title of this document was 'Developer's Manual' which indicates its purpose and origin of contents within. Ideally, a 'User Manual' is derived from the contents of this document.

QCAN2 is the successor of QCAN, implemented with modern processors, and with powerful, updated RF technology. The device maintains connector and serial port compatibility, and similar command response to its predecessor.

Much like the legacy QCAN; this device implements several modes of operation. It will act as :

a.) Intersection controller,
b.) Fixed Equipment / Door controller,
c.) Intersection and/or door controller combo,
d.) Dispatch processor, and
e.) Command repeater or Communications Repeater.

In the QCAN2 we managed to resolve most - if not all - of the challenges of its precursor. For instance with modern processors we've got more memory, we can have more contenders at an intersection. We set this limit to 255, which is a reasonable compromise to allow us to allocate static buffers.

We have also been able to improve RF communication, using a peer to peer RF broadcast subsystem. This permitted every unit to act as a natural forwarder. The RF broadcast also enabled us to maintain RF state tables across devices, which is instrumental in determining vehicle priority.

With this new RF / CPU technology, we can use hardware features to create a queue of priorities on a first come first serve basis. This priority resolution is improved, as it is coordinated by the device's operating system (semaphore) functionality.

We also added two powerful simulation software suites. One that is a successor of the QCANCom program, which sends commands to the QCAN2 serial port. The simulation then observes the QCAN2 responses.

The other simulation is a Visual Modeling program, that simulates AGV action on screen. The on screen ‘virtual AGVs’ obey the QCAN2s instructions, much like its real counterpart. The simulation later got extended to loop out to a real RS-232 serial port. The responses are then interpreted. The on-screen AGV simulates the actions of the physical vehicle. The simulation accurately models the requirements of the AGV control. It has been an instrumental tool to create a Protocol and set of State Machine states that are immune to RF disturbances, and other anomalies. It also allowed us to visually troubleshoot resolution priorities.

New tools are added. One can set up a QCAN2 to monitor all QCAN2s in range, see them in action, and optionally log the events as they happen. This is delivered as the utility called ‘AirMon’, and also installed on the laptop. The utility is developed in python, which lends itself to easy propagation into any other language. Here, one can visualize a monitoring tool delivered to the end user.

This document contains some legacy descriptions. However, most chapters introduce new features, new utilities, new tools. This document also explains concepts related to implementation details. These new concepts arose empirically, during implementation. For example, the ‘Whatsup’ process allowed us to monitor every RF table in range, and create reports and error prevention actions based upon the collective content.

Other sources of information:

- *The original ‘Developer’s manual*
- *The QCAN2 debug command line help system*
- *The Web pages of the configuration screens*

This part of the page is left intentionally blank

Implementation Commonness

QCAN2 maintains electrical interface compatibility, communication level compatibility and protocol level compatibility.

Item	QCAN	QCAN2	Notes
Terminal Strip	OK	OK	Same Size, Same Pin-out
Serial Data	OK	OK	Byte compatible
Serial Interface	RS-232 DB25	RS-232 DB9	Adapter included

Implementation Differences

Wherever appropriate, we deployed new technologies for greater functionality and more reliable operation. The table below highlights the subsystems that offer the same functionality but different implementation.

Item	QCAN	QCAN2	Notes
Configuration	Jumpers	Web Based	More reliable, contact-less operation
Radio subsystem	900 MHz Multi Channel	LORA RF, 2.4 GHz	Larger range, standards compliance
Logging	None	Event Logs	Offers Traceability
Simplified Setup	Jumpers	Setup less operation	Zero Configuration

Setup

QCAN2 features a modernized configuration and user interface. It has several configuration modes.

- WiFi / Browser, using a Cell Phone, Tablet Device or Computer; Chromebook, iPhone
- Command Terminal configuration (like putty)
- Traditional configuration mode

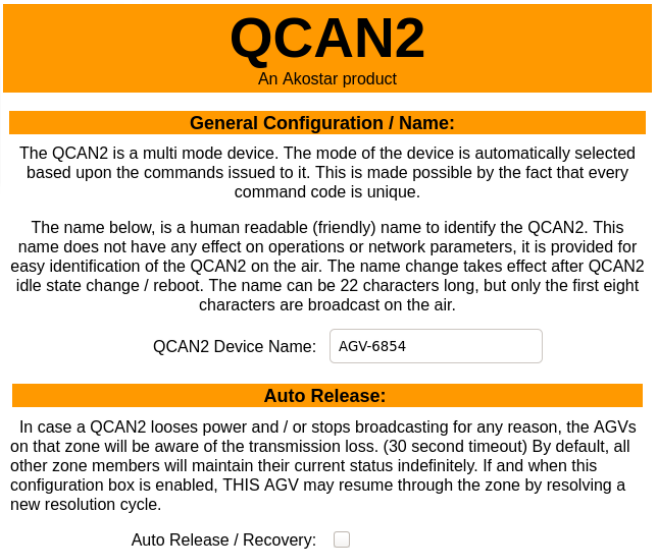
Dip Switch - less implementation

Configuring the QCAN2 via the web interface is extremely convenient. The QCAN2 will expose itself on the WiFi name space as “QCAN2-NNNN” (without the quotes), where NNNN is the last four digits of the QCAN2’s MAC address. The WiFi password is pre-set to ‘12345678’ (no quotes) The QCAN2 will listen on this interface for one to two minutes after power up, then the WiFi interface goes dormant.

Configuration main page

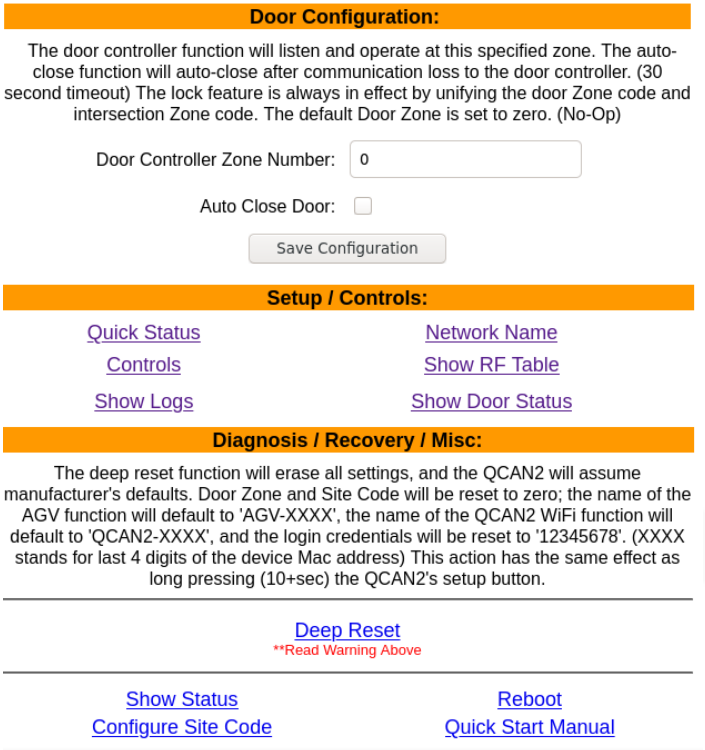
To the right, is a screen shot of the initial page the QCAN2 displays when connected to.

On this page, the AGV name can be changed. The name is advisory, as it does not effect operational parameters.



The auto release will allow this AGV to resume after any QCAN2 unexpectedly goes silent. The timeout for resume is 30 seconds. If this checkbox is not checked, the AGV holds it previous status indefinitely.

Configuration Items



The following items (left) can be configured on the QCAN2:

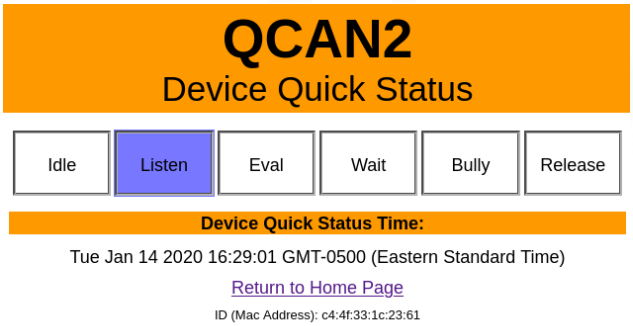
Door Zone. This is the zone the door controller listens to. The default is zero, which means no door controller function is active.

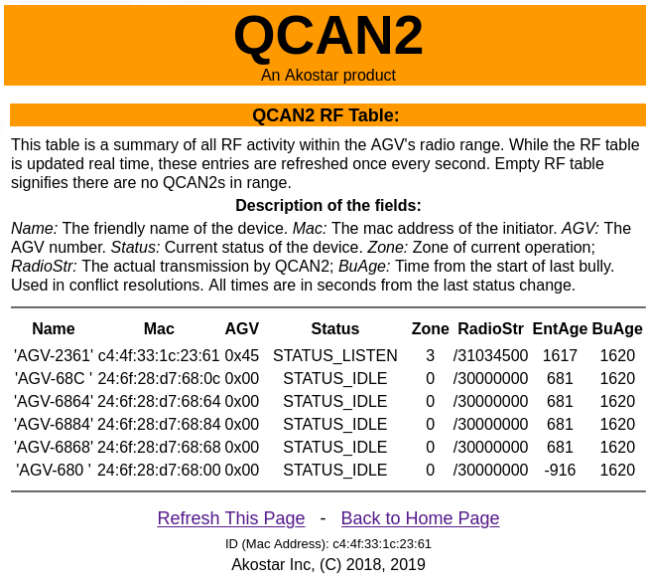
Controls. Open / Close Door; Start / Stop AGV. This control is advisory, instructions from RF override it.

Network Name: Configure WiFi Network parameters.

Show Logs:	Logging feature. Example log: 1343 QCAN_STAT_IDLE 493 1343 QCAN_STAT_RELEA 4 492
Show RF table:	List RF communication details visible by this QCAN2
Quick Status:	Show this AGVs state machine state.

Informational Items





The device status represents the current state machine state of the AGV intersection.

The RF table is a snapshot of the RF as this device sees it.

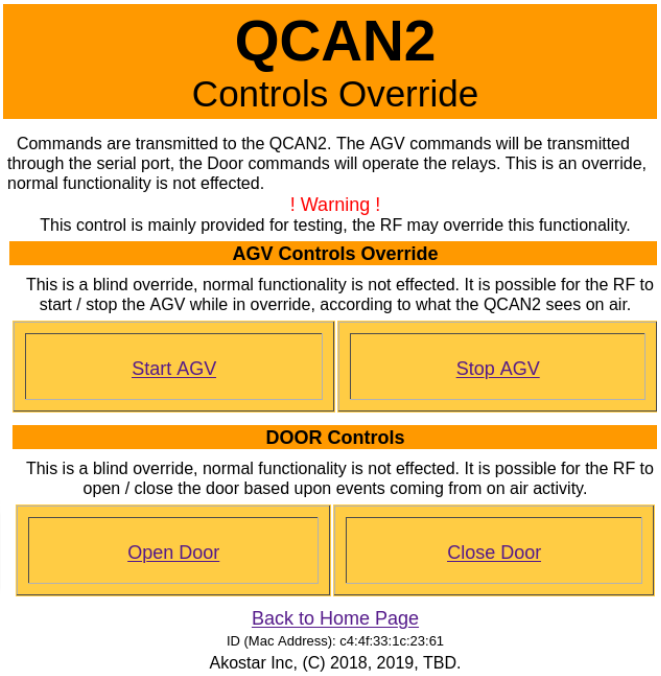
Both Statuses and Tables are live.

Control Items

The QCAN2 can be controlled from the WiFi interface. The AGV will receive start stop commands from the web interface without the priority resolution mechanism.

Same is true with the door open / close function. The web interface acts as an override, and it can be utilized when an override is required. For instance in case of stoppages or door testing.

During development we found it useful to quickly identify which QCAN2 we are interfacing with by operating the relay, and listening for the mechanical noise.



Button codes:

The QCAN2 has a push button accessible under the front panel. The button can interpret several combinations of press codes.

Press code	Description	Function
Single click		Wake up device's Wifi function
Double click		No function
Triple(3) click	Like the mouse double click, but 3 clicks instead of two	Reboot device.
Quad (4) Click	Start AGV pairing	With this mode one can pair an AGV remote. I signal arrives in 20 seconds, the pair is successful.
Penta (5) Click	Start intersection auto test mode. Four clicks in rapid succession. (like the mouse double click, but four clicks)	With this mode one can simulate serial commands without external equipment. See 'Auto test function' below.
Long press	Press and hold for 12+seconds, then release.	Manufacturer's reset. All zones are cleared, QCAN2 defaults to AGV mode.

Auto test function

With the intersection auto test mode one can simulate serial commands without external equipment. The simulated serial commands auto cycle every 10+ seconds simulating the AGV idle / listen / occupy cycles. The cycles then repeat indefinitely. The timing of the cycles is randomized, so the test is mimicking real life scenarios. The zone of the auto test is set to 100. When this test is executed on multiple QCAN2s, they show intersection negotiation. The QCAN2s LEDs should show the negotiation process, with only one AGV allowed (Green) an any one time. His test can also be used as a range test, testing the compound of radio range + negotiation range.

The auto test is exited by subsequent quad click or power cycle / reboot. This will restore normal operation.

WiFi Configuration Safety

The WiFi password can be changed from the configuration page to prevent unauthorized access to the AGV configuration. From the same screen WiFi name can be changed, the WiFi name, as it appears on the air. The WiFi configuration times out after two minutes, so the configuration WiFi cannot be initiated after that time period. Pressing the QCAN2s on board button, or restarting the unit will activate the WiFi. If the WiFi password is lost, long pressing the on board button (12+s) will reset the Manufacturer's configuration. (The pass will reset to: 12345678)

WiFi Device Compatibility

The QCAN2 Configuration pages will operate in all common platforms and browsers. We tested PC / Apple / Unix / Android / Chrome Book devices, all of which showed complete compatibility. We tested Firefox / Chrome / Safari / Edge, and it showed visually identical page results.

Zero Configuration Options

The QCAN2 – wherever possible – has a zero configuration option. For example, the AGV sends its identity on most commands. The QCAN2 can decipher that, and store this as its host identity or target zone. The zero configuration options are possible because the command codes are function specific.

The QCAN2 can distinguish all AGV actions sent to it, and bases its response upon it. On the RF side, the zero configuration is possible with the MAC address, as that is guaranteed to be globally unique.

Zero Configuration Notes

For safety, the QCAN2 will refuse to take on a second identity. (Obsolete: This function is only active in legacy mode.)

In situations where the QCAN2 is connected to a different AGV than its original host, recovery is simple. Executing a long reset (hold QCAN2 button for 12+ seconds) on the QCAN2 will clear its memory, and it will be ready to adopt its new host. *This way, on most instances, one can install a new QCAN2 without any tools or configuration just by simply plugging it in.*

Zero configuration does not apply to the door controller. The door zone has to be explicitly configured. Two methods of configuration: from the web interface, and the command line interface.

On the web interface, on the main screen, set desired zone for the door, and press the “Save Configuration’ button.

On the command line interface, the command ‘zone’ is available for setting the AGV’s own door zone. (An alias ‘doorzone’ is provided to verbalize the outcome better)

WARNING!

The door zone has to be unique to the site. If more than one door controller is operating on the same zone, the radio will receive and feed information from both zone controllers. This is why the QCAN2 that detects a second zone controller on the same zone, will continuously blink all three ‘traffic’ lights. (Red/Yellow/Green)

If there is a site door conflict, simply configure a different door zone. Alternatively, one may configure a different site code. However, when changing the site code, all QCAN2s have to be changed to that same site code. *It is recommended to keep the default site code.*

This part of the page is left intentionally blank

Event Logging

In the spirit of traceability and troubleshooting, QCAN2 maintains extensive logs. Every state machine transition is logged. Anomalies in RF Transmission / Reception are logged. Duplicate bully resolution, dead RF entry are all in the logs. The log can be accessed from the terminal command line, and from the log configuration web page. Additionally, the QCAN2 has a log host mode, where it logs everything that could be relevant to troubleshooting.

There are several levels of logging implemented:

a.) Errors;
b.) Warnings;
c.) Notices;

Below is an example of logged items in a sequence. The first field is the time stamp, the second field is the last two digits of the MAC address, and the rest describes the state transition. Interspersed lines are from the supervisory process. (This screenshot is from an older version of the code, the later versions have 4 digit MAC printout and some other minor differences)

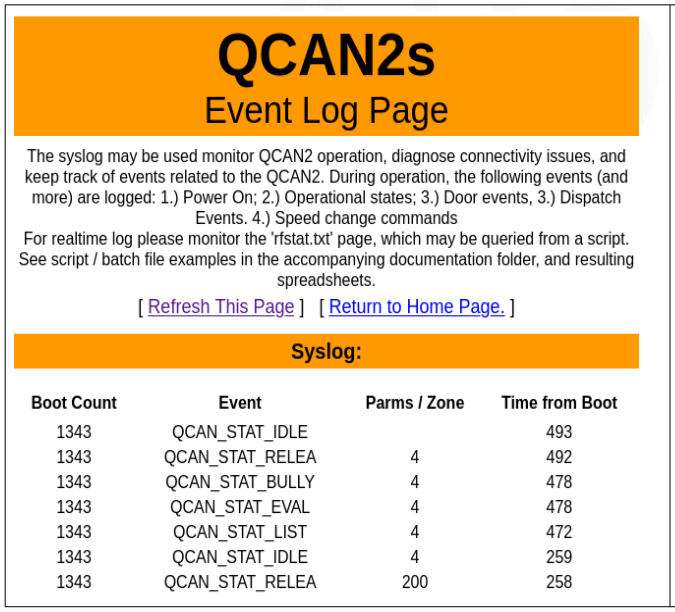
```
02:47.5 34 QCAN_STATUS_IDLE -> QCAN_STATUS_LISTEN zone=10 (0, 0)
Whatsup: Setting slow state: eval_lim 4
02:54.0 34 QCAN_STATUS_LISTEN -> QCAN_STATUS_EVAL zone=10 (0, 0)
02:55.6 34 QCAN_STATUS_EVAL -> QCAN_STATUS_WAIT zone=10 (0, 255)
Whatsup: Setting bully state: eval_lim 3
02:57.8 34 QCAN_STATUS_WAIT -> QCAN_STATUS_BULLY zone=10 (0, 0)
03:02.2 34 QCAN_STATUS_BULLY -> QCAN_STATUS_RELEASE zone=10 (0, 0)
03:04.0 34 QCAN_STATUS_RELEASE -> QCAN_STATUS_IDLE zone=0 (0, 0)
03:09.3 34 QCAN_STATUS_IDLE -> QCAN_STATUS_LISTEN zone=10 (0, 0)
Whatsup: Setting slow state: eval_lim 3
03:15.4 34 QCAN_STATUS_LISTEN -> QCAN_STATUS_EVAL zone=10 (0, 0)
03:17.2 34 QCAN_STATUS_EVAL -> QCAN_STATUS_WAIT zone=10 (0, 255)
Whatsup: Setting bully state: eval_lim 6
03:21.2 34 QCAN_STATUS_WAIT -> QCAN_STATUS_BULLY zone=10 (0, 0)
03:25.4 34 QCAN_STATUS_BULLY -> QCAN_STATUS_RELEASE zone=10 (0, 0)
03:27.0 34 QCAN_STATUS_RELEASE -> QCAN_STATUS_IDLE zone=0 (0, 0)
03:32.5 34 QCAN_STATUS_IDLE -> QCAN_STATUS_LISTEN zone=10 (0, 0)
Whatsup: Setting slow state: eval_lim 4
03:39.0 34 QCAN_STATUS_LISTEN -> QCAN_STATUS_EVAL zone=10 (0, 0)
03:40.6 34 QCAN_STATUS_EVAL -> QCAN_STATUS_WAIT zone=10 (0, 255)
Whatsup: Setting bully state: eval_lim 4
03:45.0 34 QCAN_STATUS_WAIT -> QCAN_STATUS_BULLY zone=10 (0, 0)
```

Logging Example:

The log is visible from the QCAN2s logging page. The displayed items are :

- Boot Count - The number of boots (power ups)
- Event - The event that triggered the log
- Parameters / Zone - The zone the event happened – or relevant parameters

- Time from boot - The time elapsed from last boot in seconds



QCAN2s

Event Log Page

The syslog may be used monitor QCAN2 operation, diagnose connectivity issues, and keep track of events related to the QCAN2. During operation, the following events (and more) are logged: 1.) Power On; 2.) Operational states; 3.) Door events, 3.) Dispatch Events. 4.) Speed change commands
For realtime log please monitor the 'rfstat.txt' page, which may be queried from a script. See script / batch file examples in the accompanying documentation folder, and resulting spreadsheets.

[Refresh This Page] [Return to Home Page.]

Syslog:

Boot Count	Event	Parms / Zone	Time from Boot
1343	QCAN_STAT_IDLE		493
1343	QCAN_STAT_RELEA	4	492
1343	QCAN_STAT_BULLY	4	478
1343	QCAN_STAT_EVAL	4	478
1343	QCAN_STAT_LIST	4	472
1343	QCAN_STAT_IDLE	4	259
1343	QCAN_STAT_RELEA	200	258

The QCAN2 has no knowledge of real time, but the time of the event can be calculated from inferring the time of last boot up (shift change or event with power cycle) plus the seconds from the table added.

The entries in the table are in reverse chronological order; the example to the left shows an AGV approaching and occupying zone 4. Noteworthy to see that the AGV was leaving zone 200, which is a self test zone. (Self test procedure described elsewhere in this document.)

There are additional means to see the log, described later in this document. Every QCAN is capable of reading all the RF events, and the events can be saved to a file. See: ‘Creating Logs’ later in this document.

This part of the page is left intentionally blank

Built in Command Interpreter

The QCAN2 USB port is equipped with a command interpreter. The commands can be used to simulate operations, simulate failure modes, configure QCAN2, set operational states and query QCAN2 status. The serial terminal connection is available via the USB connector next to the serial port.

The QCAN2 USB port can be used as a debug port, as a Control port, as an AGV port and as a Dispatcher port. There is no settings to configure, as the USB port automatically switches to AGV / Dispatch mode if it recognizes a valid QCAN2 packet. (/00000000\r)

To connect to the QCAN2 USB port use the following configuration:

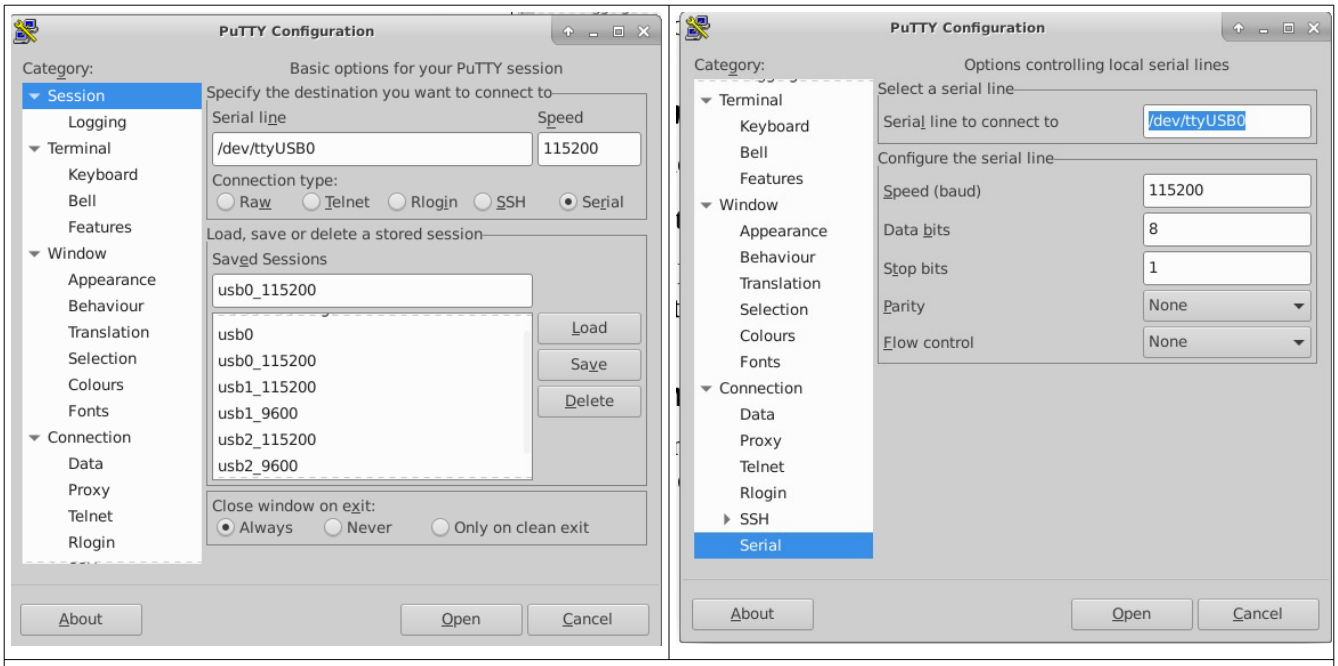
Baud: 115200 8N1 XON/XOFF disabled

The Command Line

Connecting the USB port to a computer establishes a serial connection. [most computers have the USB drivers already pre-loaded, but if not, they are publicly available; the QCAN2 uses CP2102 USB chip-set]

Upon connection, the serial port is allocated by Windows. For example, the PC sees the device as COM6. One can confirm which port is allocated in the Device Manager. Once the COM port is visible, any terminal program may be used to communicate with the QCAN. We used the open source program ‘putty’. (Installed on the demo laptop)

Configuring putty.



The screen shots above show a typical configuration for connecting to the QCAN2. The baud rate is 115200, the serial parameters are 8N1 XON/XOFF disabled.

Once connected, the QCAN2 is ready to operate from the command line. It issues a prompt in a form of ‘QCAN2>>’ (without the quotes)

Typing help at the QCAN2 command prompt will deliver a list of commands. Typing help <command> will deliver a short introduction to the command. For instance, to set the door zone, you may use the command: ‘zone’ (no quotes) To set the door zone to 5, use: ‘zone 5’ An alias is provided to make it more intuitive, the alias is called … ‘door’

It is worthy to mention that the firmware was developed on a Linux Host, but every operation is tested on windows as well.

Commands Short List

The following commands (and more) are recognized:

"id" "stat" "zone" "ls" "dump" “rfmon” …

The 'help [command]' command will deliver information about the specified command, for example 'help stat' will describe the stat command.

Command Short List Details

The commands below we used extensively, so it is described here. For more information see the detailed command descriptions later in the document.

id	Show the identity of this QCAN2
stat	Display current status of QCAN2 state machine, MAC address, and auto status.
conf	Show QCAN2 current configuration
zone [zz]	Set the QCAN2's own zone, alias for doorzone or fezone – zz=new zone; zone optional, if not specified QCAN2 will show the current zone
ls	List RF table entries
rfmon	Monitor the RF transmissions, print to the command line
xrfmon	Monitor the Transmit side of RF transmissions, print to the command line. This command theme of 'x' prefix stands as an abbreviation for 'TX", an multiple commands have 'x' versions of them.
ser	Monitor serial port transmissions, print to the command line
dump	Display current RF table, as seen by this particular QCAN2
force	Force QCAN2 into BULLY state. This simulates the error condition on a non reactive QCAN2. The test was done to follow the code path of recovery from a condition what we call a "Duplicate Bully" Use the "auto" command to resume the testing operation.
auto	The auto command will cycle the QCAN2 states on a randomized timer. With multiple QCAN2-s running, this simulates intersection traffic. This function is available as a quad click on the QCAN2's push button. TESTING ONLY

The interpreter is connected to a dumb terminal. (via USB Serial Converter) It has minimal editing key facilities. The Backspace key operates as expected, and the up arrow key recalls the last command. In case of an erroneous entry, press the Enter key to start a fresh command line. The QCAN2 will re-issue its prompt: "QCAN2 >>"

Full command list

The following list was created on Jul 29 2020 from a device terminal session.

```
ls dir bt conf stat version ver verbose macs id name zone door crzone remon remid
remdis remcomp remsin can xcan forwmon xforwmon repeat xrfmon check ddump dump
stop start ? help clear dellog showlog cpu nvs mem m reboot deepreset aclose
```

```
forward ant req rel auto azone force rfmon doorcom serial xserial stale switch
relay stoprf leak
```

Use help [function_name] for more details on specific functions.

Many of commands involve testing. For instance the command ‘auto’ will start an intersection simulation on zone 200. This was used during development, but can also be used as a tool for QC, or a tool for field test without any additional hardware.

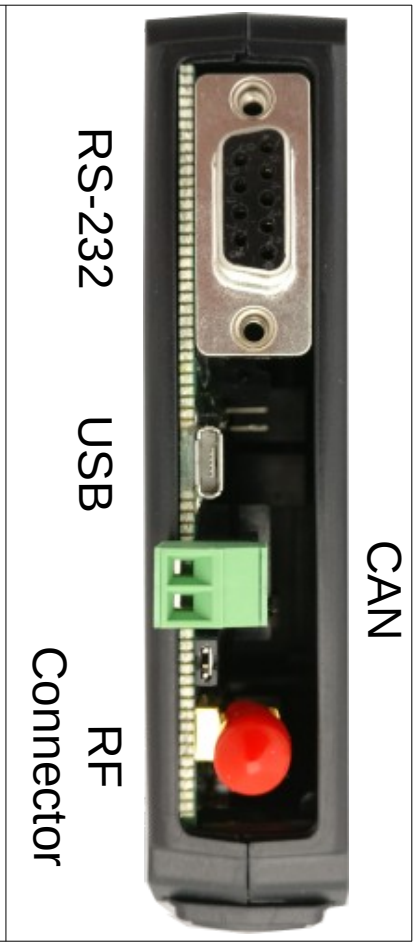
If the QCAN2 is set to an unexpected mode, as simple reboot will restore default state, default operation. One may use the command ‘reboot’ to facilitate that. If the QCAN2 has been misconfigured, long pressing the top button for 10+ seconds will reset the QCAN to its manufacturer’s defaults.

Command Example:

An example command line:
QCAN>> stat
The output of the stat (short for status) command listed below:

```
QCAN2 version 1.15
States:
  Mainstate: Q_STATUS_IDLE zone=0 par1=0x0 par2=0x0
  Doorstate: Q_STATUS_IDLE zone=0 par1=0x0 par2=0x0
  Dispstate: Q_STATUS_IDLE zone=0 par1=0x0 par2=0x0
  CR state:  Q_STATUS_IDLE zone=0 par1=0x0 par2=0x0
Configs:
  Forwarder:   0    Repeater:        1
  Autoclose:   1    Autorelease:     1
  Door Zone:   2    Cent. Res. Zone: 3
  Legacy Mode: 0    Cent. Res. Tout: 0
Properties:
  Mac:        c4:4f:33:1c:23:61
  AGV Name:   AGV-2361
  Last Mon:   686578       Boot count: 517
Diagnostics:
  rfmon:    0        xrfmon:   0
  forwmon:  0        xforwmon: 0
  canmon:   0        xcanmon:  0
  sermon:   0        xsermon:  0
  doormon:  0
  Serial Send / Recv: 27183 / 0  Serial Bad Packets: 0
 Auto step status: 0  Leak test: 0

QCAN2>>
```



This output was captured during development, but as the output has evolved, more fields are added. Please see device output for the up to date format.

Command Help Details

The following list describes all the commands in no particular order; However for the initial setup, and basic diagnosis a couple of commands are sufficient. They are: zone; stat; conf; ver; crzone; ls;

Command list:

switch: show switch states;

relay: [rel_num rel_state] set / show relay states;

stale: show inactive (stale/dead) QCAN2 table entries;

version: show QCAN2 version number;

serial: show serial recv activity;

xserial: show serial send activity;

doorcom: show door communications activity;

stat: show QCAN2 internal tables and statistics;

cong: show QCAN2 internal configuration;

bt: show QCAN2 local bully table;

check: sanity check for internal use, also, automated QC.;

ls: list (dump) current RF table;

dump: dump current RF table;

ddump: dump current door table;

stop: issue STOP command to AGV;

start: issue START command to AGV;

help: show help on command;

clear: clear/rebuild RF table and statuses;

leak: continually monitor free memory amount (for testing);

nvs: [prot] show QCAN2 NVS memory details;

mem: show QCAN2 memory consumption (leak detection);

cpu: show QCAN2 cpu consumption (diagnostic: load trace);

log: [start_point] show QCAN2 logs);

logdel: delete QCAN2 logs (disabled, use web interface);

reboot: reboot QCAN2 device;

deep: deep reset QCAN2 device. Restores manufacturing defaults;

ant: [zone_num] instruct QCAN2 with anticipate intersection;

req: [zone_num] instruct QCAN2 with request intersection;

rel: [zone_num] instruct QCAN2 with release intersection;

force: toggle to bully state with no consideration (for testing);

id: print QCAN2's identity; mac address and firmware version;

remon: toggle Remote Call enable flag (default=0) [OFF];

remdis: toggle Remote Call dispatch enable flag (default=0) [OFF];

remid [id_num]: Get / Set remote id. Range 0 .. 255 (default=0) [OFF];

remcom: toggle remote output compatibility flag;

remsin: toggle remote output single flag;

azone [none_num]: get or set zone for auto (test) functions (default=100);

aclose: toggle both autoclose / autorelease;

forward: toggle QCAN2 forwarder mode (default=0) [OFF];

zone [zone_num]: get or set this QCAN2's FE zone (Door Zone)\n

Range: 0 .. 255; 0 -> FE function off; (default=0);

crzone [zone_num]: get or set this QCAN2's CR (Central Resource) zone. \n

Range: 0 .. 255; 0 -> CR function off; (default=0);

door [zone_num]: get or set this QCAN2's FE zone (alias for zone function);

name [new_agv_name]: get or set this QCAN2's AGV name. (Friendly Name);

auto [delay]: (toggle) Q2 cycle. QCAN2 states cycled:\n

ant -> req -> rel - > delay -> ant .... ;

verbose: [ver_num] get or set verbosity; Range: 0-8\n

0 -> silent ... 8 -> noisy; (default=1);

macs: list known macs in RF table;

repeat: toggle repeater mode [mini network mode] (default: ON);

forwmon: print forwarded RS-232 data to terminal;

xforwmon: print forwarded RF data to terminal;

rfmon: print incoming RF data to terminal;

xrfmon: print outgoing RF data to terminal;

can: print incoming CAN data (diagnostics);

xcan: print outgoing CAN data (diagnostics);

stoprf: toggle RF STOP state (!! For testing only !! disables RF);

Informative sign on banner

This is what the serial port (on the USB side) shows on bootup.

```
QCAN2 Compile Time: (BUILDDATE) 21/09/20 11:25

 =-------------------------------------------------=
     QCAN2 Copyright (C) Akostar Inc. 2019, 2020.
 =-------------------------------------------------=

QCAN2 1.5, Built on Tue 15.Sep.2020 ready for action.

Mac:        24:6f:28:d7:68:5c

FE Zone:     5   CR Zone:     0
Rem ID:      0   Rem Disp:    0
SER Cache:   0   Deadline:    0
Q2 Mode:     0   Forwarder:   0
Boot count:  754  Packet size: 20  Last Mon:    878

AGV Name:   AGV-685C

Type 'help' (or '?') for a list of QCAN2 commands.
Dumb terminal; Backspace OK, Key-Up recalls last entry, Tab
Completes.

Use Ctrl-C to abort one command, Ctrl-] to abort session.

QCAN2>>
```

Split config screen to config / status:

Grouped information based upon its origination. Status and configuration.

Status screen printout:

```
QCAN2>> stat

=-------------------------------------------------=
   QCAN2 1.5 Copyright (C) Akostar Inc. 2019, 2020.
 =-------------------------------------------------=

States:
```

```
  Main state:  Q_STA_IDLE zone=0 par1=0x0 par2=0x0
  F.E. state:  Q_STA_IDLE zone=0 par1=0x0 par2=0x0
  Disp. state: Q_STA_IDLE zone=0 par1=0x0 par2=0x0
  CR state:    Q_STA_IDLE zone=0 par1=0x0 par2=0x0

Properties:
  My Mac:       24:6f:28:d7:68:5c
  AGV Name:     AGV-685C

Quick stats:
  Uptime:        903063 ms -- 0 days, 00:15:03.063
  Boot count:    754       Last Mon:      903068 ms
  RF  (Rec):     34977     RF  (Tr):      10877
  RF Err (Tr):   0         RF Skip (Tr):  0
  RF Repe (Tr):  1       SER (Rec):     0
  SER (Tr):      0

QCAN2>>
```

Config screen printout:

```
QCAN2>> conf
QCAN2 version: 1.5

Configs:
  Forwarder:    0    Repeater:     0
  Autoclose:    1    Autorelease:  1
  FE. Zone:     5    CR. Zone:     0    CR. Timeout:  0 ms
  Ser. Cache:   0    Ser. Dline:   0 ms
Remote Call:
  Rem.Call On:  0    Rem.Call ID:  0 (RCB)
  Rem.Disp On:  0    Compat Flag:  0 (RDISP)
  Single Flag:  0
Diagnostics:
  rfmon:        0    xrfmon:       0    forwmon:      0
  xforwmon:     0    canmon:       0    xcanmon:      0
  sermon:       0    xsermon:      0    doormon:      0
Stats:
  SER   Send / Recv: 218673 / 0  -- Serial Bad Packets: 0
  RF    Send / Recv: 11737 / 37730
Diag. Status:
  Auto step status: 0
  Leak test status: 0
```

The stat command output:

```
QCAN2>> stat
 =--------------------------------------------------=
   QCAN2 1.4 Copyright (C) Akostar Inc. 2019, 2020.
 =--------------------------------------------------=


States:
  Main state:  Q_STA_IDLE zone=0 par1=0x0 par2=0x0
  F.E. state:  Q_STA_IDLE zone=0 par1=0x0 par2=0x0
  Disp. state: Q_STA_IDLE zone=0 par1=0x0 par2=0x0
  CR state:    Q_STA_IDLE zone=0 par1=0x0 par2=0x0
Properties:
  My Mac:       bc:dd:c2:f1:99:41
  AGV Name:     AGV-9941


  Curr Time:    189700770 micro sec.


Quick stats:
  Boot count:   4452    Last Mon:     189705
  Uptime:       189709 ms -- 0 days, 00:03:09.709
  Packets RF:   3859
  Packets SER:  0
```

RF Safety, Coexistence

The QCAN2 has two radios. One for configuration, one for RF negotiation, the main radio. The configuration radio only operates for a short period on power up. The main radio operates continuously, and updates it surroundings with information about the state of the system. All this is collected in the virtual RF table, residing in every QCAN2.

The main radio uses LORA (Long Range or Low Radiation) technology, which allows the longer range without extra power. The LORA technology affords superior performance, low interference, and in our testing it proved to be a very reliable transmission medium.

RF security considerations.

The RF table is updated to reflect the current state of the systems within range. The payload of the update is protected by a cryptographically secure hash, and if it detects a corrupt packet it will not propagate it into the RF table. The payload is not encrypted, so external utilities can monitor the state of the system (AirMon utility example provided)

Introduction, Terminology

Terminology, definitions:

AGV	Automated Guided Vehicle
Host AGV	The AGV the Q-CAN is connected to
Contender AGV	The AGV that is a contender for the shared resource (FE/Door/Intersection ...)
Serial Port	All three port options: RS232, CAN, USB
RS232 Port	The specific port, RS-232
RF	Radio Frequency; in this context the QCAN2 radio subsystem
Serial Messages	Messages from the RS232 Serial Port or CAN bus or USB port
Commands	The Q-CAN radio and the vehicle computer communicate with several types of commands. A command is repeated as a series of messages until the desired reply is received.
Dispatcher	A computer system that can send work orders to a Q-CAN AGV when it is in a predefined zone and requesting dispatching.
Dispatching	Sequence of controls and commands used to send work orders to Q-CAN AGV.
Door (Bridge)	A place in the system where a vehicle needs to control, or interface with, fixed equipment in its path.
Intersections	Places in the system where vehicles cross each others path, and have the potential to collide.
Messages	There is a continuous flow of messages between the Q-CAN radio and the vehicle computer. If either unit stops sending messages the other will attempt to stop the vehicle.
Q-CAN	Quick Configurable AutomatioN mostly written as QCAN2
Q-CAN System	The collective of all Q-CAN devices on a single site.
Q-CAN Blocking	System Q-CAN Blocking System using small, limited range, radios to control AGV access to intersections.
Repeater	Radio Device for extending the useful range of the RF communications
Forwarder	Device / QCAN2 Mode to forward from RF to RS-232 and vice versa
Stations	Places in the system where the vehicle can stop to load or drop.
IDRD	Collective identifier for the the Intersection Controller, the Door Controller, the Repeater, and the Dispatch.
Door mechanism	The system that responds to door open commands including signaling door state.

XXX	Item not known or named at the time of writing.
IPPC	Intersection Pier to Pier Controller
CR	Central resource controller

General Description

Most of this chapter is a derivation from the original spec.

The messages are carried between the AGV and QCAN2 via a serial port. In the QCAN2 the serial port could be RS232, CAN or USB. In the description below the numbers assumed to be are hexadecimal numbers, and the characters appear as they would in the 'C' language.

The general format of the messages:

Meaning	**Start Char**	**Message ID**	**Zone**	**Data_1**	**Data_2**	**Carriage Return**	**Notes**
Typical Usage	/	Command	Zone Number	VID	Status	\r	
Example1	/	01	10	03	00	\r	Send
Example2	/	81	10	03	00	\r	Recv

Status Bit descriptions

This is the format of data_2 field. It is used in many different contexts, so the meaning of the bit depends on which subsystem is in operation. Most of the bit allocation is identical to the previous specification version, except Bit_2; also for completeness, I included the switch bits.

The status bits have changed from the original specification as per approval of Mr. Hinman. The progression of this particular bit field modifications: a.) Mar 4 2020; b.) Apr 9 2020; Changed as per recommendation from per Mr Hinman c.) Jun 01, 2020

Bits, Authoritative as of Jun/02/2020:

Bit Number	**Bit Value**	**Short Name** (name in code)	**Description**
Bit_7	0x80	BLOCK_QCAN2	There is some other vehicle radio in the zone. AGV stops or

			slows based on context.
Bit_6	0x40	IN_CONTROL	The radio has uncontested control of the zone.
Bit_5	0x20	ERROR_QCAN2	Error Occurred; Missing Spec on error source
Bit_4	0x10	RELAY_BIT1	Set when Relay 2 is ON
Bit_3	0x08	BLOCK_FIXED	Fixed Equipment Blocking zone
Bit_2	0x04	RELAY_BIT0	Set when Relay 1 is ON
Bit_1	0x02	DOOR_BIT1	Fixed EQ input_1
Bit_0	0x01	DOOR_BIT2	Fixed EQ input_0

Status code bit allocations (as updated)

Bit_7 (0x80) when set => Block (stop or slow / blocked)
Bit_6 (0x40) when set => This AGV is in control of the zone
Bit_5 (0x20) when set => Error occurred
Bit_4 (0x10) when set => No AGV occupies this zone

Bit_3 (0x08) when set => Fixed equipment occupied
Bit_2 (0x04) when set => No AGV present in the zone << NEW

Bit_1 (0x02) when set => Switch_1 active (FE context only)
Bit_0 (0x01) when set => Switch_0 active (FE context only)

The rationale for bit_2 is to show if any AGV is in this zone either as an occupier or as a requester. This allows the AGV to see all that is present in the zone, which in turn allows for an informed choose on the part of the AGV for alternate route.

QCAN2 BIT Allocation (jun/2020)

```
#define BLOCK_QCAN2   BIT_7
#define IN_CONTROL        BIT_6
#define ERROR_QCAN2       BIT_5
#define RELAY_BIT1        BIT_4
```

```
#define BLOCK_FIXED         BIT_3
#define RELAY_BIT0          BIT_2
#define DOOR_BIT1           BIT_1
#define DOOR_BIT0           BIT_0
```

Serial Communications

AGV to QCAN2 and QCAN2 to AGV

The data format is identical to the legacy QCAN. The command, Zone, Data_1, Data_2 are ASCII Hexadecimal characters, from 0-9 and A-Z. (Please note, in the current Savant serial com test suite, lowercase characters are not accepted)

Start	Command	Zone	Data_1	Data_2	Carriage Return
/	00-FF	01-FF	00-FF	00-FF	CR

An example of communication string is:

/01237800\r

Which translates to command: 0x01 (anticipate) zone: 0x23 AGV: 0x78 parameter: 0x00

Example: Open / Request Fixed Equipment:

This is chapter is mostly a copy of the spec.

The QCAN2 responds identically to these two commands, with the only difference in the response is the command received. 83 for command 03 and 84 for command 04. The QCAN2 always responds with the state of the two inputs on the fixed equipment controller regardless of whether or not the vehicle has control of the zone. The QCAN2 will

report the state of the two inputs in the STATUS byte, input 1 on STATUS bit 0 and input 2 on STATUS bit 1. The fixed equipment controller will assert output 1, and the vehicle QCAN2 will turn the BLOCKED bit in STATUS on or off to reflect the state of the fixed equipment controller's input 1 as well as the process of gaining control of the zone.

The 1.5 second priority resolution delay will apply to the STATUS BLOCKED bit response to input 1 the same as it does to becoming IN_CONTROL. That is, the vehicle QCAN2 can respond with the STATUS BLOCKED bit false (0) for 1.5 seconds, even if input 1 is false. After the 1.5 second priority resolution delay, the STATUS BLOCKED bit should be true (1) if either IN_CONTROL or input 1 is false.

Q-CAN QCAN2 Specification Fixed equipment is permitted to ignore output 1 if the door or other fixed equipment is not supposed to be controlled by the vehicle. This could be a normally open door that we don't want the vehicle to run into if it is closed for some reason.

A fixed equipment QCAN2 is the Zone Controller. Since a vehicle could request control of the zone with the Request Zone command 02 before issuing these commands, and must not surrender control it has already gained, the fixed equipment controller must also monitor for this condition. Read Fixed Equipment (0A; 8A): The QCAN2 responds with the state of the two inputs on the fixed equipment controller regardless of whether or not the vehicle has control of the zone. The QCAN2 will report the state of the two inputs in the STATUS byte, input 1 on STATUS bit 0 and input 2 on STATUS bit 1. The state of those inputs will have no effect on the BLOCKED STATUS bit.

Once the vehicle is in control of the zone, fixed equipment controller output 1 will reflect STATUS bit 0 in the command, and output 2 will reflect STATUS bit 1. The outputs will only go active if explicitly commanded by the vehicle in control of the zone, and will be cleared when the vehicle turns them off or surrenders control of the zone. The "turn on output 1" implied in the Open / Request Fixed Equipment commands does not happen in response to this command. Likewise STATUS bits 4~7 returned by the QCAN2 reflect only the state of the requested zone, and are not affected by the state of the inputs on the fixed equipment controller.

A fixed equipment QCAN2 is the Zone Controller. Since a vehicle could request control of the zone with the Request Zone command 02 before issuing this command, and must not surrender control it has already gained, the fixed equipment controller must also monitor for this condition.

Dispatcher Commands:

(detailed elsewhere)

CAN BUS Communications

AGV to QCAN2 and QCAN2 to AGV

The data format communicated over CAN is identical to the Serial RS-232 communications format. The max length of the CAN BUS message is 8 bytes, so the 10 byte QCAN2 message is transmitted in two frames; an 8 bytes frame and a 2 bytes frame.

The CAN BUS parameters:

Parameter	**Value**
CAN Data Speed	250000
Data length	8 bytes + 2 bytes on next packet
**Transmit message ID (MSG_TID)	0x19EE5401
**Receive message ID (MSG_RID)	0x19EE5402

** Transmit / Receive as seen from the QCAN2’s point of view.

Test and communication examples are provided for the commonly used ROBOTELL USB to CAN transceiver. (on the accompanying Jump Drive)

Forwarders, Repeaters

As mentioned in the earlier section, our new RF empowered us with new features. Every QCAN2 acts as a natural forwarder. If there is an RF starvation point, or long distance communication needing a repeater, placing a QCAN2 at that position should solve the problem.

Repeater

The repeater functions without any configuration changes. This relies on a subsystem we coined as the 'mini network'. The mini network forwards the packets it sees, with an algorithm similar to how the internet works. Packets have a TTL (Time To Live) field, and this field is decremented every time the packet is forwarded. The packet is discarded if the TTL value reaches zero. Also, a packet has a Unique ID field, and packets that are seen before are not processed. This assures that a packet will travel to all destinations with the help of the intermediaries. One exception is the FE switch information. As it is updated without state change, the FE door switch information gets a unique ID every time it is updated.

Forwarder

The forwarder mode is a special mode for the QCAN2. It forwards packets arriving from the Radio Subsystem to the RS-232 port. These packets then can be transmitted via the serial port to another forwarder. The other Forwarder will broadcast the packet on the RF. This transmission is seamless, the RF packets forwarded rejoin the 'mini network'. The mini network handles the forwarded packets with the same algorithm as the normal packets; the TTL field and Unique ID assures that a packet will propagate via air or wire. When the QCAN2 is in forwarder mode, the CPU LED will double blink.

The forwarder mode can be configured on the main web page or on the command line. The command to toggle forward mode is: 'forward'

State Machine Description.

The QCAN2 'State Machine' refers to a set of states, induced by incoming events from the serial port and the RF subsystem. The state machine is basically the underlying control mechanism to achieve the desired responses. While the QCAN2 state machine is relatively complex, we attempt to describe the foundational aspects of it.

State machine definition. A good example of a state machine is a TV power button. Two states, on or off. The response of the button is influenced by the previous state of the TV; if it was off, it comes on and if it was on, it powers down.

The QCAN2 state machine attempts to implement the state transitions required to implement the QCAN2 protocol. It is significantly more complex than the state machine for the TV button, never the less, it is simple enough to implement in an embedded controller.

In implementing the state machine, we attempted to follow the specs with our wordings. The 'C' language variable names give an indication of the internal states. Here is a sampler of the state machine variable names: (these names are also printed on the terminal when operating)

QCAN_STATUS_IDLE QCAN_STATUS_LISTEN QCAN_STATUS_EVAL
QCAN_STATUS_WAIT QCAN_STATUS_BULLY QCAN_STATUS_RELEASE

The state machine transition names are coined similarly as they appear in the specification, with some additions. For instance, the QCAN_STATUS_EVAL is a transitional state, where the QCAN2 makes a decision on the next required state. Another example is the QCAN_STATUS_BULLY state, which is a result of the successful occupy test.

The ideal state machine implementation would require one to intercept code from every state transition to every other possible state. Clearly, it is not practical to do that, so the simplest way to achieve good coverage is to permit state jumps, and make most states insensitive to the previous state. This is called the 'stateless' or (mostly stateless) implementation. (the HTTP protocol is a good example of a stateless implementation)

Here is one instance of the stateless transition in the QCAN2. When 'Occupy' signal arrives without the 'Anticipate' signal, the state machine transition assumes the intent of the 'Occupy' instead of signaling the error status of a skipped state. The state machine 'warps' to the desired state. This is appropriate, as in another section of the specification 'Occupy' is specified without the prelude of 'Anticipate'.

Abrupt zone change. The QCAN2 can accommodate abrupt zone change. When the current zone changes without following the underlying procedure of release / anticipate, the QCAN2 allows the new zone to take effect. When an abrupt intersection zone change takes place, a new resolution starts. In case of the door controller (FE), a release state is injected, with possible door close, as specified.

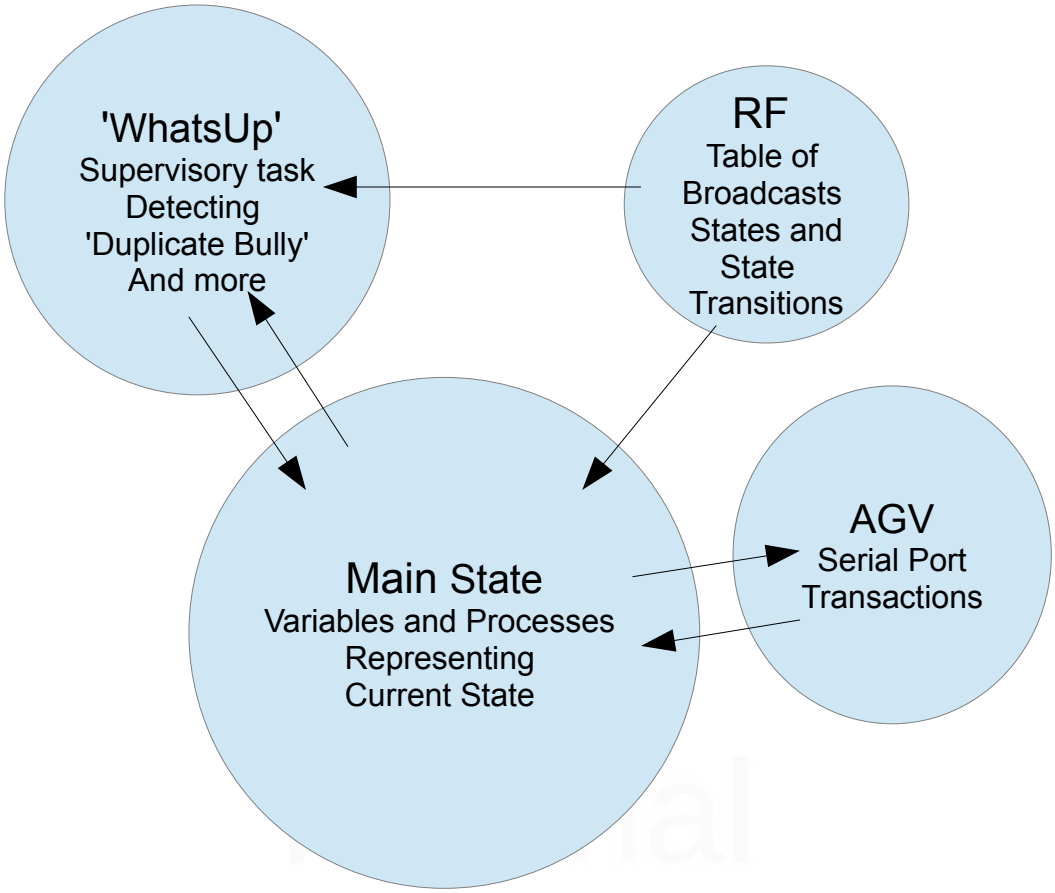
The benefit of the statelessness is that missed signals do not cause stoppages, the state machine will elect the possible intent of the sequence. This is also one of the reasons one may click around the simulation 'willy-nilly' without any ill effects.

Naturally, the statelessness has limits. Those limits will be uncovered when one tests the QCAN2. We hope that the shortcomings will be shared, so corrections can be made by us.

This part of the page is left intentionally blank

State diagrams, charts

This is a preliminary chart, only to serve as a basis for discussion.
(as of Feb-2020 – it is implemented)



Protocol State Description

These notes apply to protocol interfaces. Can be safely ignored, unless protocol level changes are implemented. It is unlikely that the protocol level data needs any change in the future, as the payload is variable length. One can extend the payload without protocol changes.

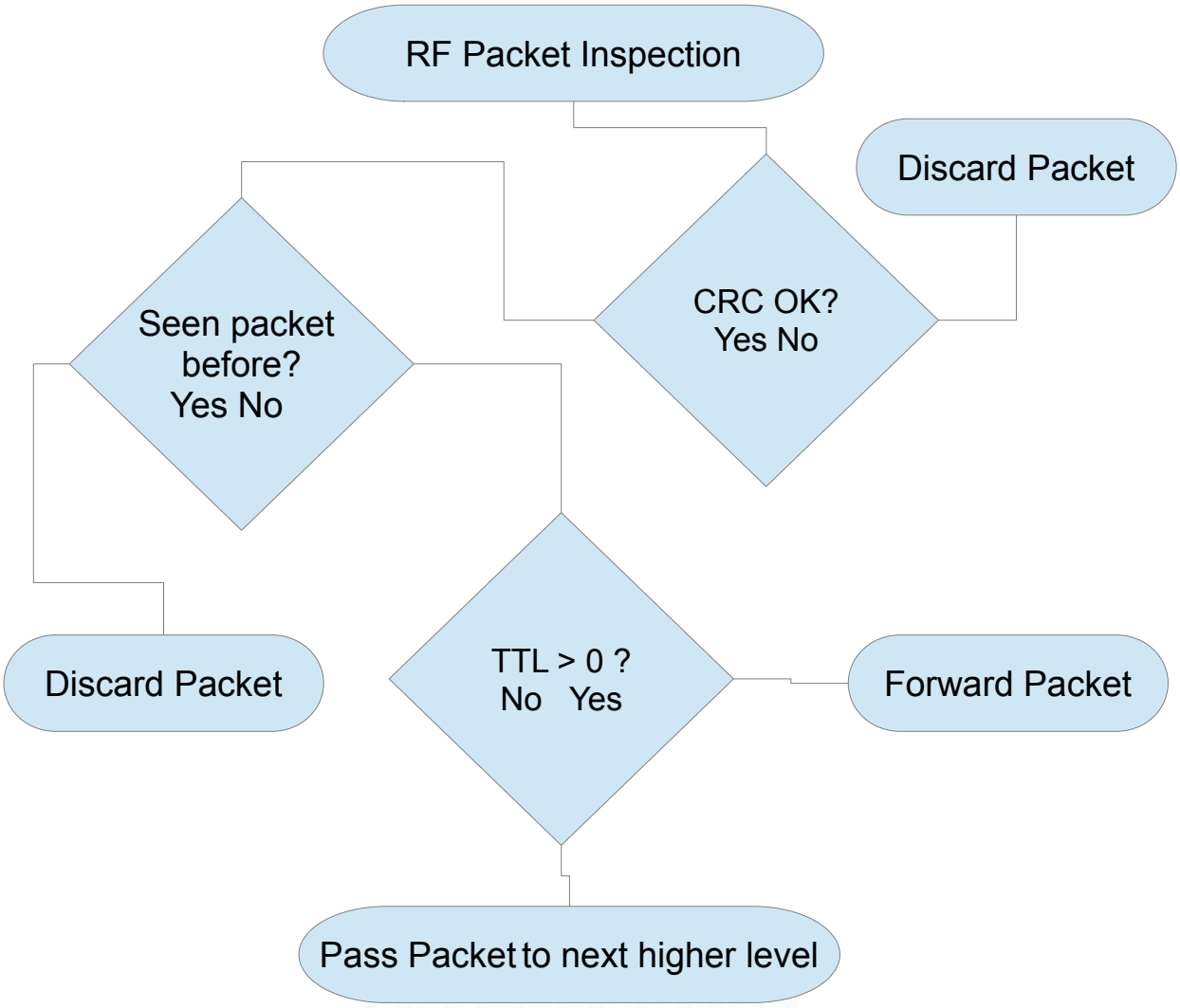
```
typedef struct _qcan_espnow_data_t {

    uint16_t  ver;                        // Indicate version of the protocol
    uint8_t   ttl;                        // Indicate packet trip count
    uint16_t  seq_num;                    // Sequence number of ESPNOW data.
    uint16_t  crc;                        // CRC16 value of ESPNOW data.
    uint32_t  packid;                     // Magic number to uniqely identify pac
    uint8_t   self_addr[ESP_NOW_ETH_ALEN];// The originator
    uint16_t  paylen;                     // Payload length
    uint8_t   payload[0];                 // Real payload of ESPNOW data.
} __attribute__((packed)) qcan_espnow_data_t;
```

```
#define PROTVERSION 0xabcd
```

As of this writing, the protocol version is 0xabcd. If you change the protocol structure, redefine the PROTVERSION, and create a code switch construct to handle the new protocol data and the old protocol data.

Just for completeness, here is how the RF data makes it into the QCAN2 RF table:



Intersection Controller

(Implemented, informational section only)

Due to the dynamic nature of the intersections, QCAN2 will resolve AGV crossing priority via radio broadcast. In general, the resolution will proceed on a first-come first serve basis. QCAN2 and the AGV will go through several phases to coordinate intersection priority.

The first phase is called '*anticipate intersection*'. In this phase the QCAN2 will listen, and determine if there is a contesting AGV at the same intersection. If there is, the QCAN2 instructs the AGV to slow it's speed by transmitting on its serial port the OCCUPIED status, setting BIT_7

The second phase is called *'request intersection'*. At this point the QCAN2 will determine if the intersection is occupied. If intersection is occupied, the AGV will be instructed by the QCAN2 to stop, by transmitting on its serial port the OCCUPIED status BIT_7. Update: see new bit allocation at the end of this chapter. If the intersection is NOT occupied, the QCAN2 will continue to respond on the serial port to the ALIVE command, so the AGV will proceed.

The third phase is called *'release intersection'*. When the AGV reaches this phase, it instructs the QCAN2 to release this zone. The QCAN2 signals all other QCAN2s that the AGV cleared the intersection. Then, the other QCAN2s enter into an evaluation phase, and the AGV that has been waiting the longest, proceeds. (First come first served)

Detailed Description of Messages:

This is an augmented version of the specification, reflecting how the QCAN2 implemented the individual commands.

Idle (0x00; 0x80):

The vehicle and QCAN2 exchanges this command and response pair when there is no task for the QCAN2 to do. This also serves a watchdog function, and the release function. The release function signals when the vehicle no longer wants to control a zone that the vehicle currently controls or is competing for. The idle function is awakened in other conditions like abrupt zone change or unexpected status change. TODO: to resolve ambiguities, separate functionality to distinct units. (release in particular)

Watch Dog Stop (0xFF):

This message originates from the QCAN2. If and when the time gap between valid Serial messages are too long. The vehicle is responding with a full stop, and waits until new, meaningful message arrive.

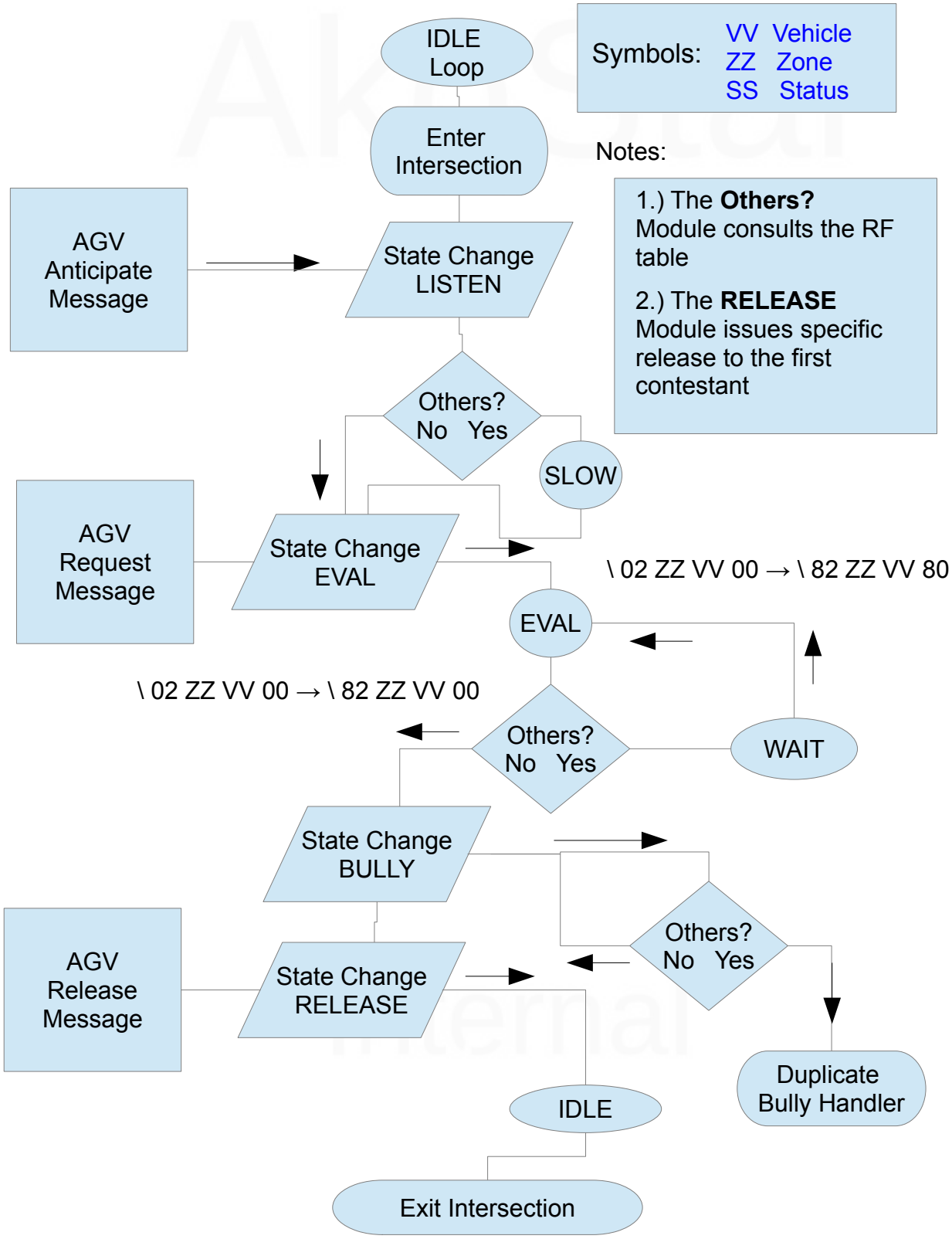
Anticipate Zone (01; 81):

A vehicle may send this command when it wants to know if another vehicle is in a zone without requesting control of that zone itself. This can be used by vehicle software to select an unoccupied route from among several possible routes to its destination. The QCAN2 will turn the BLOCKED bit in the STATUS byte on whenever it detects another vehicle competing for the zone, and off when no other QCAN2 is competing for or controling the zone. The QCAN2 may turn the BLOCKED bit on and off repeatedly as the presence or absence of other vehicles in the zone changes.

Request Zone (0x02; 0x82):

When a vehicle sends this command, it is requesting control of the zone. The QCAN2 will respond with one of the STATUS values described above under "Control of a Zone" to show where the vehicle QCAN2 is in the process of gaining control of the zone. Vehicle software will decide (possibly based on the QCAN2's default settings) how long the vehicle may continue to move without controlling the zone, to allow for priority resolution.

Intersection Logic Chart



Fixed Equipment Controller / Door Controller

The QCAN2 acts as a door controller if the FE zone number is set to a number between 1 - 255. The zone number 0 (zero) denotes that the QCAN2 is not in zone controller mode. This setting can be applied from the command line or the web interface. It is recommended to power cycle the QCAN2 after setting this mode. The mode and zone is stored in the QCAN2's non volatile memory. The mode can be set any number of times, there is no limit.

If there is another operating zone controller that is in radio range with the same zone number, the QCAN2 will blink the Red / Yellow / Green lights on off until the zone conflict is resolved.

The QCAN2 Fixed Equipment controller facilitates communication between the AGV and a Fixed Equipment (Door Controller) mechanism. The AGV communicates with the FE controller (door controller) via the designated door commands. The QCAN2 door controller has four major function groups:

a.) Occupy Door Zone / Open Door
b.) Control the door (open / close relays)
c.) Report the status of the door.
d.) Leave Door Zone, Close door

The commands correspond to functions of the door. a.) Open door b.) Close door.
The status reports of the door correspond to: c.) Door in transition, Door opened, Door closed. The status of the door is passed back to the AGV for further action.

Intersection controller and door controller combo

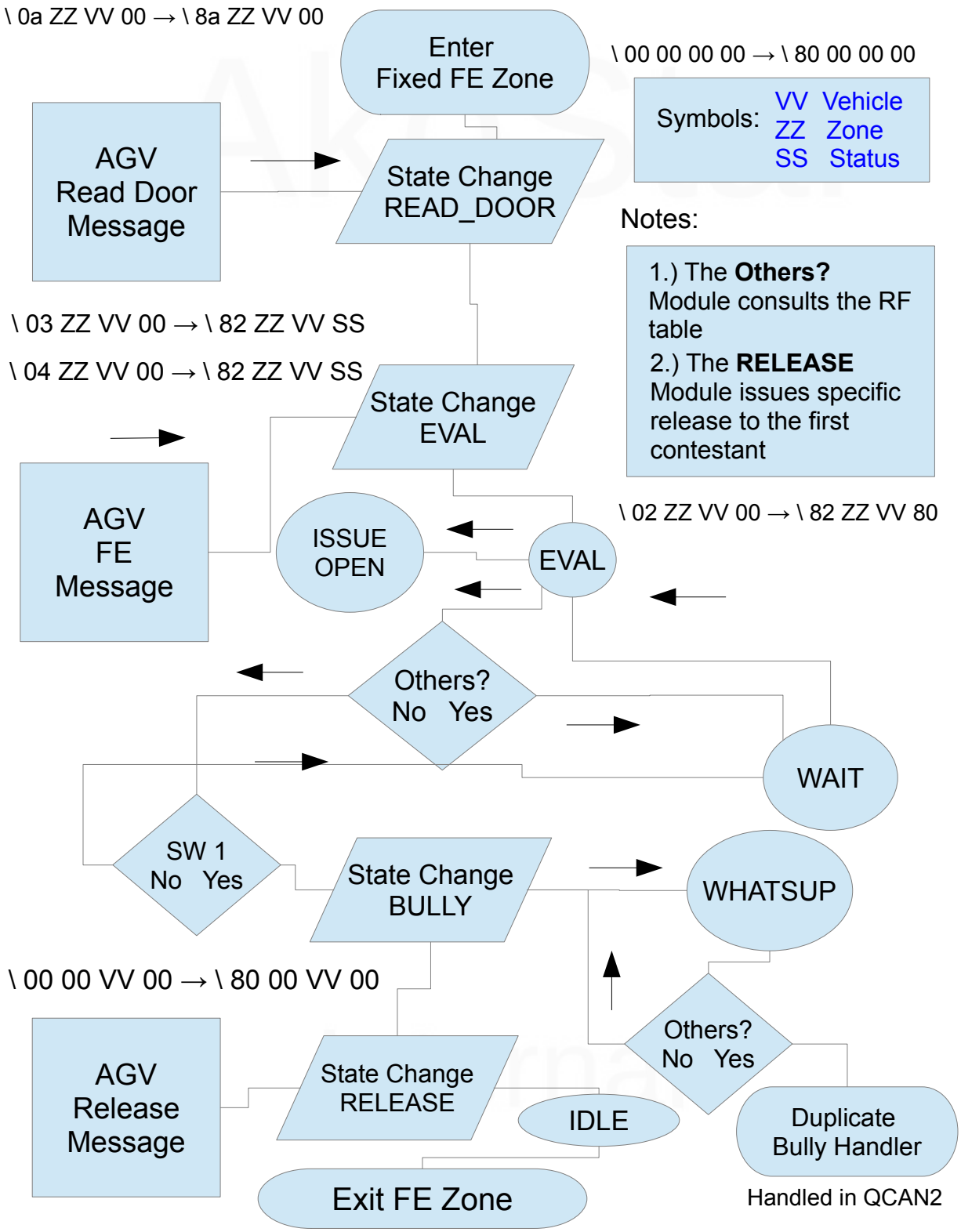
When an AGV approaches an intersection, that is also a door, the command sequence is stacked. As the door is a singular resource, the intersection controller logic may be simplified. This intersection door combo may be deployed where the peer-to-peer intersection controller has difficulty. (Future Design Requirement)

Door controller forwarding

The Fixed Equipment controller will echo all commands to its RS-232 serial port, and will obey commands coming from said serial port. Attaching a serial cable to a QCAN2 forwarder's RS-232 serial port will relay all Fixed Equipment commands in both directions. The forwarder can be placed strategically to solve RF starvation issues, for example controller behind steel door. There is also the possibility to place an RF forwarder without the use of cables.

Fixed Equipment Flow Chart

Below is a skeleton flowchart for the fixed equipment. It depicts the ideal flow, as the real life process is somewhat more complex. Over the evolution of the code the protocol changed, but the overall functionality stayed intact.


\ 0a ZZ VV 00 → \ 8a ZZ VV 00
Enter Fixed FE Zone
\ 00 00 00 00 → \ 80 00 00 00
AGV Read Door Message
State Change READ_DOOR
Symbols:
VV Vehicle
ZZ Zone
SS Status
Notes:
1.) The Others? Module consults the RF table
2.) The RELEASE Module issues specific release to the first contestant
\ 03 ZZ VV 00 → \ 82 ZZ VV SS
\ 04 ZZ VV 00 → \ 82 ZZ VV SS
State Change EVAL
AGV FE Message
ISSUE OPEN
EVAL
\ 02 ZZ VV 00 → \ 82 ZZ VV 80
Others? No Yes
WAIT
SW 1 No Yes
State Change BULLY
WHATSUP
\ 00 00 VV 00 → \ 80 00 VV 00
Others? No Yes
AGV Release Message
State Change RELEASE
IDLE
Duplicate Bully Handler
Handled in QCAN2
Exit FE Zone

Fixed equipment timings

This is a short description of the fixed equipment control protocol response timing. There are various timings for different sections.

- Switch state propagation
- Relay command propagation
- Relay state propagation

Switch state propagation:

From the change in switch state to the signal that arrives to the AGV QCAN2, the operation has several phases. 1.) Initiating switch status request; 2.) Broadcasting switch status; 3.) Injecting switch status request result to serial stream;

Relay command propagation:

From the relay change request to the signal that arrives to the FE QCAN2, the operation has several phases. 1.) Sending relay change request; 2.) Acting on the relay change request.

Relay state propagation:

From the change in relay state to the signal that arrives to the AGV QCAN2, the operation has several phases. 1.) Broadcasting relay status 2.) Injecting relay status result into the serial stream;

Each phase in the above descriptions last a communication clock tick (appx. 300 ms each)

The switch state propagation thus lasted 900 ms +- 2 ticks; This was deemed to be too long, so a caching mechanism is introduced. The SW state is broadcast on FE idle state, and the information is stored in a new table. (Much like the RF Table)

Background:

The idle time keep alive packet now contains the SW state and Relay state. The information is cached in a separate table, and delivered on demand. The door table cache is updated on a one second periodicity, and expires in three seconds. By that time the regular door SW protocol delivers information, and the cache is discarded / ignored.

SW Cache specification:

Minimum latency 0 ms; max latency 1060 ms avg latency 530 ms for the broadcast.

(60 ms de-bounce + 1000 ms periodic update)

Table size 32; oldest entry discarded if case of full table. The cache is applied to the stream immediately upon request.

This allows the AGV QCAN2 to deliver immediate SW status response.

General Timing parameters:

The timing of the QCAN2 has constant elements, variable elements and random elements. This is due the fact, that actions concerning a common resource (like FE) need to be staggered in time. That way, the likelihood of competing occurrences / events is diminished.

The QCAN2 state machine, in its simplified form consists of the following states:

State	Timing
QCAN_STATUS_IDLE	Indefinite**
QCAN_STATUS_EVAL	`EVAL_TIME + rand( EVAL_TIME / 2)`
QCAN_STATUS_BULLY QCAN_STATUS_WAIT	Indefinite** Decision based on other AGVs positions
QCAN_STATUS_RELEASE	`RELE_TIME + rand(RELE_TIME / 2) or` `FE_RELE_TIME + rand(FE_RELE_TIME / 2)`
QCAN_STATUS_IDLE	Indefinite** Loop back

Indefinite** No timing constraint, AGV signals the Beginning / End

The states in the chart above cycle from the beginning to the end and loop back to the beginning. This simplified chart contains only the case state: “all is a go”, which we call the ‘YES ROUTE’. Timing for all other cases can be derived from the above graph.

```
#define EVAL_TIME       500     // How long to wait for bully resolution (ms)
#define RELE_TIME       500     // How long to pump IN release messages (ms)
#define FE_RELE_TIME    600     // How long to pump FE release messages (ms)
#define CR_RELE_TIME    600     // How long to pump CR release messages (ms)
#define REZ_RELE_TIME   800     // How long to pump release on abrupt zone (ms)
#define DUPBULLY_TIMER  300     // The time between two bully evaluations (ms)
#define WHATSUP_TIMER   300     // The time between two whatsup queries (ms)
```

Summary of door controller timing:

The switch status shows up in sub-second timing, the relay status shows up within 1 .. to .. 2 second delay. The relay command takes effect in 0.3 to 0.9 seconds.

Remote Call Functions

This is a new feature of the QCAN2 units. The objective is to provide the remote call facility substituting / replacing / extending the remote call function with a QCAN2 device. The rationale behind this feature is to take advantage of the QCAN2-s larger range, use the added functionality that is empowered by the mini network and the added coverage afforded by the forwarder. In addition, a significant cost saving by using uniform equipment to fulfill multiple functions and simplifying logistics by stocking fewer number of items.

The remote call feature consist of two parts:

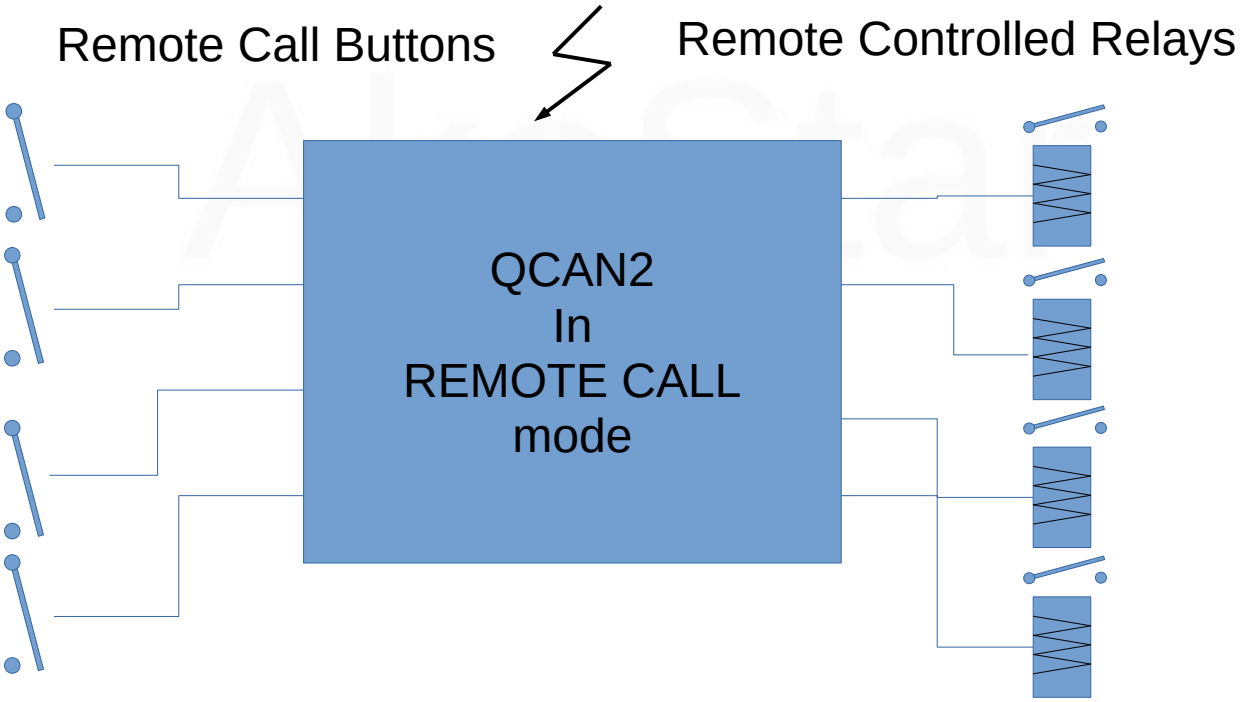
1.) The Remote Call Transmitter unit. (Remote Call Button) [RCB]
2.) The remote Call Receiver / Dispatcher / Responder unit. (Remote Call Dispatch) [RCD]

Remote Call Functions, Transmitter (RCB)

There are two versions of the call function the QCAN2 emulates:

1.) Wired buttons for fixed installation. 2.) Wireless remote, handheld installation.

The wired installation accommodates four push buttons. The button state change is transmitted to the configured Receiver. The state change transfer is in real time with the following propagation time specifications: : Button de-bounce: 80 msec Transmit latency: < 12 msec Receive latency: < 14 msec. The Button press and Receiver response delay is within human perception limits, it looks and feels ‘instantaneous’.



The Remote Call Button (RCB) mode also listens for incoming RF on its Remote Call ID. The incoming transmission that associated with this ID; will control the four relays. The same interpretation as the switches. For instance, the string ‘wxyz’ turns all relays off, ‘WXYZ’ turns all relays on. See a more detailed description on the protocol later in this document.

Remote Call Receiver Responder / Dispatch.

The receiver has two distinct modes.

1.) Remote Call Dispatch mode
2.) Remote Call Compatibility mode.

Remote Call Dispatch mode (RCD)

The Remote Call Dispatch mode coexists with the existing dispatching and AGV data streams. The RCD uses a compatible RS-232 format, all fields are arranged in the same order, and occupy the same position. Two new commands (or new op-codes) are introduced to accommodate the RCD events. One command is signaling the dispatch computer that a switch changed (button pushed / released). The other command is used to send to the Remote Call Button Unit a new relay / LED status. The commands have the following format (in ‘C’ notation):

/cczzwxyz\r\n

where:

cc	The new command (op-code) (0x20 for button event, 0x21 for relay event) The QCAN2 will set Bit 7 on all replies, so the simulator shows the opcode with bit 7 set. (0x20 - > 0xa0)
zz	The caller's identity from 0-255 (0x0 .. 0xff) This ID is also used in replying.
wxyz	switch state / relay state string. Lower case letter is open switch / open relay, upper case is closed switch / closed relay. (open=off closed=on)

Examples:

/0A02wxyz\r\n	All Switches off
/0A02WXYZ\r\n	All Switches on
/0A02wXyz\r\n	Switch 2 on

The new commands are picked to be out of range from the existing ones. The dispatch computer may be programmed to receive these op codes in conjuncture with the existing ones, and respond accordingly fulfilling both dispatch and call functions.

The QCAN2 may be connected to a computer host though a USB connector. When the USB connector receives a QCAN2 compatible instructions (for example the IDLE message), the USB communication channel switches to QCAN2 mode. (from terminal mode) This way, the QCAN2 dispatch function can be fulfilled via a simple USB connection to any PC.Call Dispatch Compatibility mode

The compatibility mode was designed to be a direct substitution for the Akostar Remote AGV call unit. The QCAN2 can be configured to output a serial stream matching the stream output by the Akostar unit. It also outputs a compatible stream to control the Remote Control Button units. The QCAN2 AGV Serial Port operates as the input / output for the remote call compatible function.

Input format: ('C' notation)

zzwxyz\r\n	zz=caller identity 0-255; wxyz = switch string

Input format, single shot: ('C' notation)

zzs\r\n	zz=caller identity 0-255; s= one of w, x, y, z = switch string letter

Output format:

zzwxyz\r\n	zz=caller identity 0-255; wxyz = relay string

Intentionally left blank

Configuring Remote Call

The Remote Call function is dormant unless it is explicitly enabled. This is to separate operations from interfering with each other. For example, a dispatch computer that might not be ready to interpret the extra op codes, may signal error when an unknown op code arrives.

The remote call consists of two major parts: the sender and the receiver. Even though both parts are capable of sending and receiving, we make the following terminology assumption:

- The sender is the Remote Call Button Unit (RCB)
- The receiver is the Remote Call Dispatch Unit (RCB)

Please make sure that only one of these functions are activated, as the RCB overrides other operations. This may not be desirable if one wants the Remote Call Dispatch function.

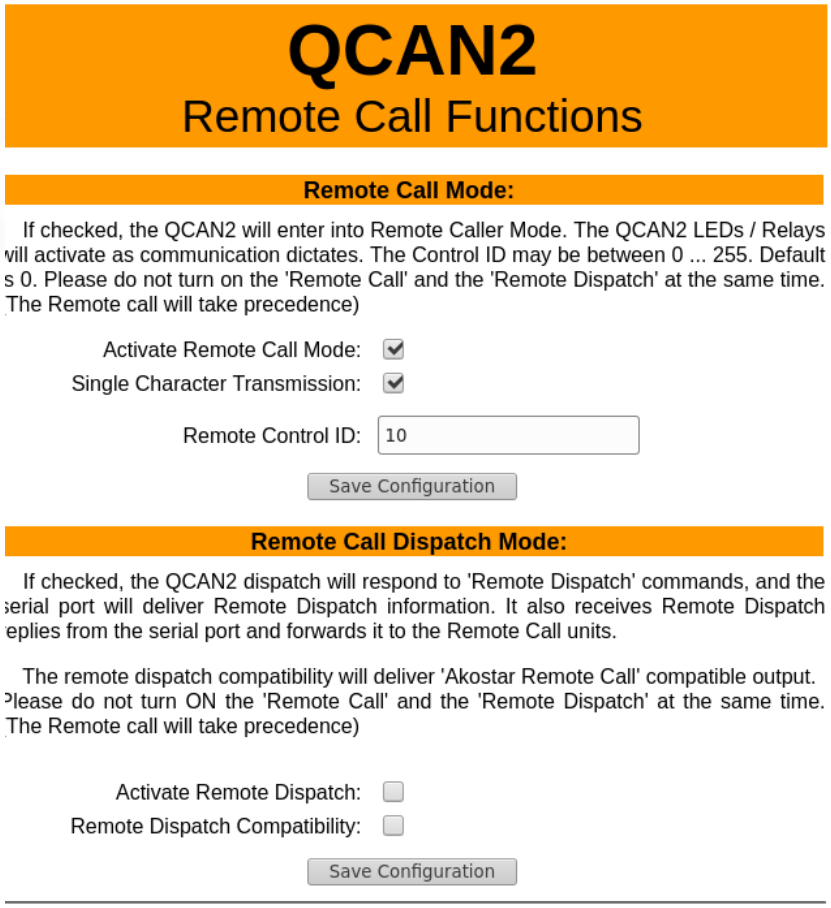
The Remote Call Functions may be configured from both the web interface and the command line. The command line commands for the remote control functions are:

remon	To toggle the enable / disable RCB mode	remid	Get / set remote id
remdisp	To toggle the enable / disable RCD mode	Remcomp	To toggle the enable / disable RCD Compatibility mode

Activating Remote Call mode will put the QCAN2 into Remote Call Button (RCB) mode. All the other QCAN2 functions will stop, this makes the unit a dedicated RCB.

Activating Remote Call Dispatch mode will put the QCAN2 into Remote Call Dispatch (RCD) mode. The QCAN2 now will inject the remote call requests into its RS-232 data stream. The injected data stream is atomic, (queued) and will not interfere / modify any other data already in the steam.

The QCAN2 Remote Dispatch compatibility will force the Remote control Dispatch to output an Akostar compatible serial stream.



Remote Control Op codes summary:

The code allocations for the RCD system is as follows:

Description	Command	Zone	Data_1	Data_2	Notes
RCD Received	0x20	00-FF	wx	yz	Upper case denotes ON Lower case denotes OFF
Send RCD Request	0x21	00-FF	wx	yz	As above

Opcodes are allocated beyond current opcode range, please give feedback if the opcode is acceptable, or revision is needed.

Deploying the Remote Call

The QCAN2 needs to be configured for the Remote Call function. On the initiator side, the RCB has to enabled, and it needs to be configured with the Remote Call Function ID.

On the receiver side, the RCD has to be enabled, and the compatibility mode chosen.

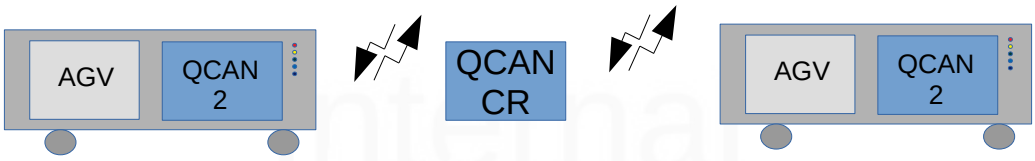
The Central Resource Controller (CR)

This is a new feature added to the QCAN2 units. The intention is to provide an option when considering intersection communication, for as pier to pier has been accomplished and while we expect no stoppages for your users, the end result is that the pier to pier depends upon the fact that all AGVs can transmit and receive, and if an AGV is coming thru a tunnel or an area of no or limited RF, then this concept of having a dedicated intersection controller offering a means for an AGV to request control of an intersection from a fixed piece of equipment will provide a positive means of preventing a stoppage by having an AGV ask permission to have control of an intersection before it takes control ~ rather than look for RF that may not exist due to environmental concerns ~ thereby with the dedicated intersection controller, the AGV will not be able to gain access to an intersection should there be lack of rf communication (hence positive control, rather than negative (for as with pier-to-pier the lack of RF results in control) and with a dedicated intersection controller as an option available to SA you now have a positive means of control wherein lack of RF results in an AGV stopping before entering an intersection.

The Central Resource Controller

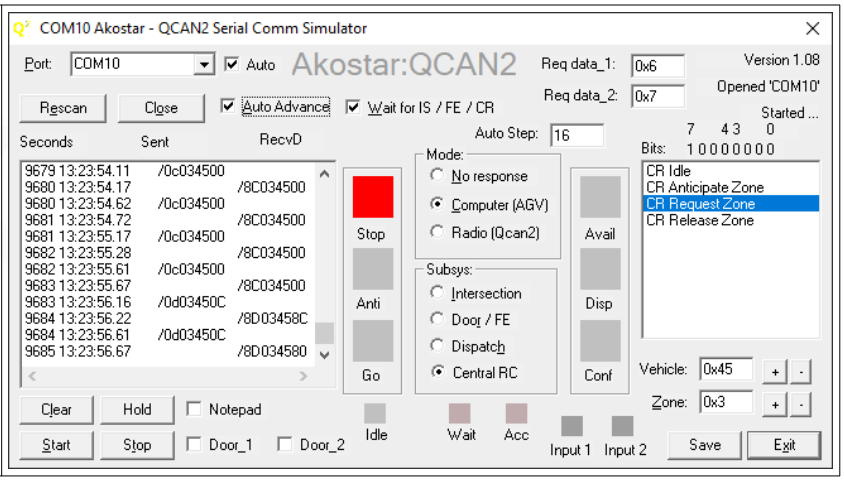
The Central Resource Controller is based upon the theory of a dedicated, zone based central controller. The resource is now controlled by a central resource controller, turning the cycle of positive acknowledgment to a positive action. If the communication to the central resource controller is severed, it is interpreted as a negative action, stopping the AGV. This assures that the AGV only goes if it has permission from the central resource controller, which results in a failure less theory of the intersection controller. This theory is now implemented in the QCAN2.

The peer to peer intersection protocol theory has an inherent weakness. The control is established peer to peer, relying on the participants to obey the controlling peer. However, if the communication to the controller is severed, the other peers interpret it as a go signal. The controlled resource relied on positive acknowledgment for a negative action.



In order to facilitate the Central Resource controller, we added new opcodes to the serial protocol. The new opcodes operate on a similar process theory as the existing protocols. They have a similar bit allocation for the same fields. The Zone; the AGV; etc .. all occupy the same bytes. The only exception is that they are driving the CR subsystem. To keep compatibility with the existing specifications, the other opcodes operate as before, unchanged.

The simulator already contains the opcode drivers, and it contains an added radio button labeled “Central RC”. We updated the wait logic to behave the same way as the other subsystems, the block states wait indefinitely until non-block state is encountered, than stepping resumes until the next blocked state. This can be used for unattended testing.



New Op codes

The code allocations for the CR system is as follows: *IPPC = Intersection Pier to Pier Controller

Description	Command	Zone	Data_1	Data_2	Notes
CR Idle	0x00	00	00	00	Same as IPPC*
CR Anticipate	0x0c	01-FF	00-FF	00-FF	Same as IPPC Command 0x01
CR Request	0x0d	01-FF	00-FF	00-FF	Same as IPPC Command 0x02
CR Release	0x0e	01-FF	00-FF	00-FF	Explicit release

CR Corner cases, power downs.

When the power to the central resource controller goes down, all the AGV QCAN2s preserve their previous state. The currently authorized vehicle keeps going, the stopped vehicles stay stopped. This is to assure that the intersection clears out in case of power interruption to the FE controller.

When the power to the AGV QCAN2 goes down, or Serial Communication to the QCAN2 is interrupted, a 10 second timeout counter is started. If the power / communication is restored before the timeout expires, operation resumes uninterrupted, states are preserved as before the interruption.

If the power / communication stays off longer then the timeout, the AGV QCAN2 releases the zone in the Central Resource Controller, operation on that zone may resume.

Configuring the CR controller. The web page is extended to have a text box with the CR zone field, and a timeout field. The timeout defaults to zero, resulting in no timeout action.

How it all fits together: The new serial command op codes introduced may be used as they are currently implemented. It is presented as a separate subsystem, so the AGV implementer has free range on using Peer to Peer or CR or any combination thereof.

A good option is adding them to the AGV's control logic. Something along the line of CR resolution and Peer to Peer resolution stacked as dependents. This would take care of any door / intersection combo.

If that method is impractical, we could merge the commands 0x03 and 0x04 into the CR functionality.

Implementation details: the CR identifies the AGVs (QCAN2s) by their MAC address, which is guaranteed to be globally unique. The permission to any particular QCAN2 is identified by the last four

digits of the MAC address. This is unique, considering the head part of the MAC address is chip manufacturer specific.

LED-s related to the CR controller

When the CR has an actively authorized AGV, the (metaphoric) traffic LED lights all light up. All three lights, the Red, the Yellow and Green are on for the duration of authorization. The lights go off when the AGV releases the CR zone. If there is another AGV competing for that CR, the lights come back on shortly thereafter. The average time gap is 300ms to 600 ms; he re-authorization takes place rapidly.

Rationale for the CR Controller.

The CR is not offered because the other protocols have a weakness. It is offered to create a solution for corner cases, completing the target task without any allowance for weakness in the underlying protocol. We do recognize the extra effort on installing and deploying the the Central Resource Controller(s), however, the added expense creates a future proof solution. Also, a single QCAN2 can serve both as CR controller and FE controller at the same time. (with almost no limitations)

Deploying the CR controller

One configuration item is needed. The CR zone. This can be configured from the web page, or the command line. A single command sets the zone. Use: crzone <zoneNumber>. As usual, the command line offers help by typing: ‘help crzone’ (without the quotes).

Dispatcher

Communication with the dispatch. The dispatcher can request an AGV to fulfill a task. To dispatch an AGV, the AGV issues the following commands:

a.) Waiting Dispatch
b.) Accept Dispatch.

To dispatch an AGV, the dispatcher issues the following commands:

a.) Anticipate Vehicle
b.) Request Vehicle
c.) Confirm Vehicle

Dispatcher Commands:

Below is a verbatim quote from the spec:

Messages 05 through 09 are dispatching commands. Message 07 (Anticipate Vehicle) is sent by the dispatcher computer when it has not yet established a link to a specific vehicle, so it is not passed on to any vehicle. When any other dispatcher command is received by either a vehicle or dispatcher radio with an established link, it is passed unchanged from one computer to the other. When a radio first receives a dispatcher command, it will respond to the computer with an echo of the command with the 80 bit added to the command ID.

Once the radios have established a link, messages are passed back and forth with minimal processing by either radio until the link is broken. The link is broken when either radio receives some command that's not a dispatcher command, changes the selected zone, or the dispatcher computer sends the Anticipate Vehicle message 07 to its radio. A dispatcher radio could be an Zone Controller. Since a vehicle must request control of the zone with the Request Zone command 02 before issuing these commands, and must not surrender control it has already gained, a dispatcher radio acting as a Zone Controller must monitor for this condition.

Dispatch Data format:

Name	**Prelude**	**Command**	**Zone**	**Vehicle**	**Status**	**Postlude**
Abbreviation	/	cc	zz	dd	ss	\r
Name				**Vehicle**		
Abbreviation				vv		
Name				**Data_1**	**Data2**	
Abbreviation				aa	bb	
Name				**Disp #**	**Req / Conf**	
Abbreviation				pp	qq	

Aliases: aa=data_1 bb=data_2 vv=vehicle tt=target pp=dispatcher 00=zero oo=optional data qq=request/confirm

AGV Command Waiting Dispatch” (05):
AGV Command “Accept Dispatch” (06):
Dispatcher Command "Anticipate Vehicle" (07):
Dispatcher Command "Request Vehicle" (08):
Dispatcher Command "Confirm Vehicle" (09):

QCAN2 Dispatch flow table + Implementation details:

(This table is incomplete, though the implementation is believed to be complete)

Client (AGV)				**Server (Dispatcher)**			
Serial (prefix: /)		**RF (prefix: &)**		**Serial (prefix: /)**		**RF (prefix: &)**	
Challenge	**Response**	**Challenge**	**Response**	**Challenge**	**Response**	**Challenge**	**Response**
05 zz vv 00	85 00 00 00	05 zz 00 00	05 00 00 00	00 00 00 00	80 00 00 00	00 00 00 00	00 00 00 00
05 zz vv 00	87 00 pp ss	07 zz 00 00	07 00 pp 00	07 00 00 00	85 00 00 00	05 00 00 00	05 00 pp 00
05 zz vv 00	88 00 aa bb	05 zz 00 00	08 00 aa bb	08 00 00 00	85 00 00 00	08 00 00 00	05 00 aa bb
06 zz vv 00	88 00 aa bb	06 zz 00 00	86 00 00 00	08 00 00 00	86 00 00 00	06 00 00 00	08 00 00 00
06 zz vv 00	89 00 aa bb	00 zz 00 00	00 00 00 00	09 00 00 00	89 00 00 00	00 00 00 00	89 00 00 00
00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00

Below is a verbatim quote from the spec:

Waiting Dispatch (05):

The vehicle radio can only accept this command if the vehicle is already in control of the zone. The vehicle signals that it is looking for commands from a dispatcher by sending this command to its radio. The vehicle must supply its VID as Data_1. The message will be passed unchanged to the dispatcher radio when a link has been established. The radio will respond with ID 85 if it does not receive a command from a dispatcher radio, otherwise it responds with the dispatcher message unchanged.

Accept Dispatch (06):

The vehicle radio can only accept this command if the vehicle is already in control of the zone. The vehicle signals whether or not it can accept the command in a Request Vehicle message 08 with this message. It must supply its VID as Data_1 and the destination as Data_2 if it can accept the command. It must replace the destination number in Data_2with FF if it cannot accept the command for any reason. This data is passed unchanged to the dispatcher.

Dispatcher Command "Anticipate Vehicle" (07):

The dispatcher computer sends this command to its radio when it is looking for a vehicle to command. Data_1 and Data_2 are zero because the dispatcher doesn't know yet what vehicle will control the zone. The radio will respond with ID 87 if there is no vehicle Waiting Dispatch in the zone. The radios will pass on the Waiting Dispatch message 05 from the vehicle when there is a vehicle in the zone sending that command to its radio. Dispatcher Command "Request Vehicle" (08):
The dispatcher computer sends this command when it has received the Waiting Dispatch message 05 from a vehicle, or it has received Accept Dispatch from a vehicle after telling the vehicle there are more commands for that vehicle. Data_1 holds the desired destination and Data_2 holds application-specific command bits. This data is passed unchanged to the vehicle.

Dispatcher Command "Request Vehicle" (08):

The dispatcher computer sends this command when it has received the Waiting Dispatch message 05 from a vehicle, or it has received Accept Dispatch from a vehicle after telling the vehicle there are more commands for that vehicle. Data_1 holds the desired destination and Data_2 holds application-specific command bits. This data is passed unchanged to the vehicle.

Dispatcher Command "Confirm Vehicle" (09):

The dispatcher computer sends this command in response to the Accept Dispatch message 06 from the vehicle. Data_1 holds the vehicle ID. Data_2 can hold one of these three values: 00: commands the vehicle to end the dispatching process and proceed to its next destination. FE: commands the vehicle to clear its destination queue and start the dispatch process all over. FF: commands the vehicle to request an additional destination from the dispatcher. This data is passed unchanged to the vehicle. The dispatcher computer sends this command until it receives the "idle" response (ID 89), or it receives another Waiting Dispatch message 05.

This part of the page is left intentionally blank

Repeater / Forwarder

QCAN2 has an advanced packet forwarding mechanism. We coined the term 'mini network', because it operates on a similar principle as the internet. Packets are repeated from QCAN2 to QCAN2, until the time to live field reaches zero. This makes for a very efficient data transmission, and guarantees data will reach every single QCAN2 installation.

The repeater will hold the signal that it receives, and repeats it. Three modes.

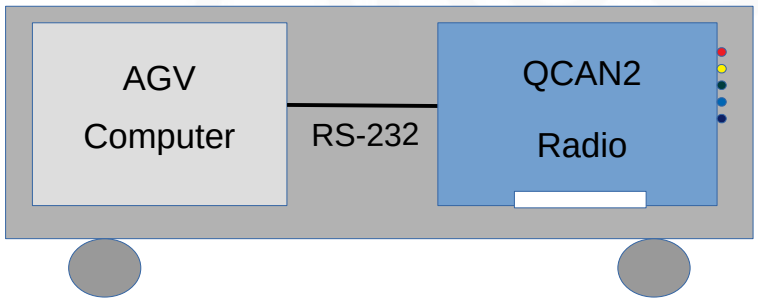
a.) Wired to Wireless
b.) Wireless to Wired
c.) Wireless to Wireless.

By default the Wireless to Wireless repeater is always on. For most cases, placing a QCAN2 at the point of radio signal starvation should permit improved communication.

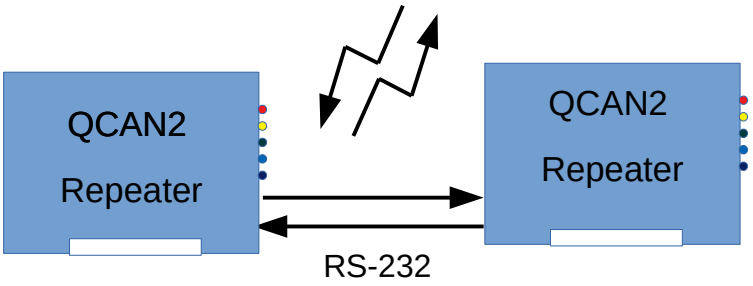
The Mac address can also then be observed as the AGV number, (or a virtual zone number, or intersection number.) Because the QCAN2s communicate with each other based on this Mac address, zero configuration is automatic. The other major aspect of the configuration-less operation of the QCAN2, is that every control command is unique, and pertinent to a specific function, so the QCAN2 will always be able to distinguish what action to perform. One exception: the QCAN2 needs to be told it is a Wired to RF repeater. This can be done on the main page of the web configuration.

(This is addressed further in the door controller / intersection controller combo)

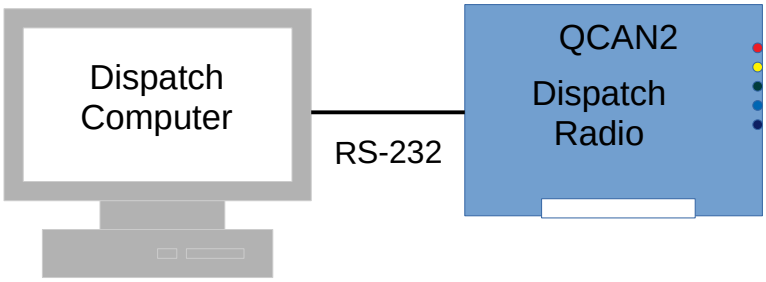
Propagation Algorithm: Packets have unique ID and Time To Live field. They are repeated in both direction, except when duplicate ID is encountered or the TTL == 0. Repeater connection: RS-232 9600 Baud 8N1 Max. 100 – 150 meters (328 – 500 feet) on CAT 5 (or like) cable. Both RF traffic and serial traffic is repeated, yielding excellent coverage in all circumstances.



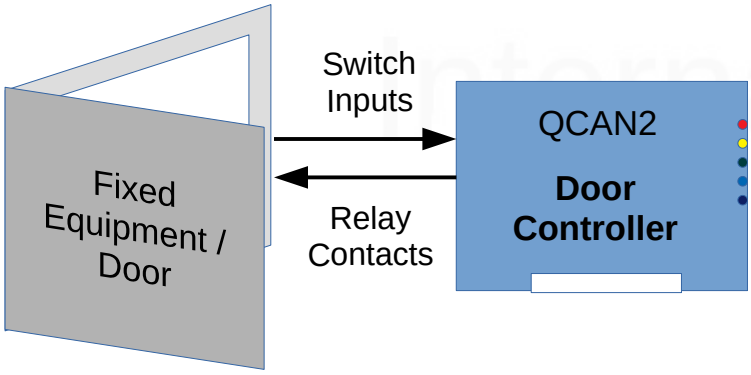
QCAN2 Radio to any other QCAN2 and any Repeater. From Repeater to any other QCAN2



All packets are replicated on both Serial and RF, subject to the above algorithm.



All packets are replicated by every QCAN2 and by all Repeaters, subject to above algorithm.



All packets are replicated bidirectionally, as above.

Creating RF Logs

In troubleshooting the QCAN2, it is useful to have a log of QCAN2 activities. The QCAN2 provides a dynamic snapshot of the RF Table when the file ‘rfstat.txt” is accessed from it via the web interface. Simply connect a PC to the QCAN2 WiFi Interface, and load the file into a browser or use the open source ‘wget’ from the command line. (http://gnuwin32.sourceforge.net/packages/wget.htm)

While this file can be seen from the browser, true usefulness reveals when accessed from a script. One can create a log file of all activities by continually querying the RF Table, and appending it to a file.

Below, a LibreOffice (**OpenOffice) import of the created text file. The contents of the file mirrors the RF Table at the time of capture, indexed by time. The first column is QCAN2 processor time in seconds. Any event’s time can be calculated from referencing the script start time from the second line, with the time of event. [**LibreOffice the import was achieved with two clicks, OpenOffice asked for several options]

	A	B	C	D	E	F	G	H	I
1	SysTime	Name	Mac	LastCom	T1	T2	T3	RFstr	
2	# Started log at: Wed 02 Oct 2019 02:59:50 PM EDT								
3	643:	'agv_chris'	'30:ae:a4:1a:aa:34'	'QCAN_STATUS_IDLE'	0	731	682	737	/30000000
4	643:	"	'3c:71:bf:16:d0:40'	'QCAN_STATUS_BULLY'	1	736	736	738	*34014500
5									
6	644:	'agv_chris'	'30:ae:a4:1a:aa:34'	'QCAN_STATUS_LISTEN'	1	738	682	738	/310145FF
7	644:	"	'3c:71:bf:16:d0:40'	'QCAN_STATUS_BULLY'	1	736	736	739	*34014500
8									
9	645:	'agv_chris'	'30:ae:a4:1a:aa:34'	'QCAN_STATUS_LISTEN'	1	738	682	738	/310145FF
10	645:	"	'3c:71:bf:16:d0:40'	'QCAN_STATUS_BULLY'	1	736	736	740	*34014500
11									
12	647:	'agv_chris'	'30:ae:a4:1a:aa:34'	'QCAN_STATUS_LISTEN'	1	738	682	738	/310145FF
13	647:	"	'3c:71:bf:16:d0:40'	'QCAN_STATUS_BULLY'	1	736	736	741	*34014500
14									
15	648:	'agv_chris'	'30:ae:a4:1a:aa:34'	'QCAN_STATUS_LISTEN'	1	738	682	742	/310145FF
16	648:	"	'3c:71:bf:16:d0:40'	'QCAN_STATUS_BULLY'	1	736	736	742	*34014500
17									
18	649:	'agv_chris'	'30:ae:a4:1a:aa:34'	'QCAN_STATUS_LISTEN'	1	738	682	742	/310145FF
19	649:	"	'3c:71:bf:16:d0:40'	'QCAN_STATUS_RELEASE'	0	743	0	743	*35000000
20									
21	650:	'agv_chris'	'30:ae:a4:1a:aa:34'	'QCAN_STATUS_LISTEN'	1	744	682	744	/31014500
22	650:	"	'3c:71:bf:16:d0:40'	'QCAN_STATUS_IDLE'	0	744	0	744	/30000000
23									
24	651:	'agv_chris'	'30:ae:a4:1a:aa:34'	'QCAN_STATUS_BULLY'	1	746	682	746	*34014500
25	651:	"	'3c:71:bf:16:d0:40'	'QCAN_STATUS_IDLE'	0	744	0	745	/30000000
26									
27	652:	'agv_chris'	'30:ae:a4:1a:aa:34'	'QCAN_STATUS_BULLY'	1	746	682	747	*34014500
28	652:	"	'3c:71:bf:16:d0:40'	'QCAN_STATUS_IDLE'	0	744	0	746	/30000000
29									
30	653:	'agv_chris'	'30:ae:a4:1a:aa:34'	'QCAN_STATUS_BULLY'	1	746	682	748	*34014500
31	653:	"	'3c:71:bf:16:d0:40'	'QCAN_STATUS_IDLE'	0	744	0	748	/30000000
32									

The script (below) is delivered with the QCAN2’s prototype package, and it is quoted below for reference. It is native to Linux, but a Windows version is provided as well. (see mon_qcan2.bat)

```
#!/bin/bash
#
# Script to monitor QCAN2 status, and create a logfile
# Written for Linux, but same programs are available for windows.
# Results are in rflog.txt
#
LOGFILE=rflog.txt
echo "SysTime, Name, Mac, LastCom, T1, T2, T3, RFstr" >> rflog.txt
echo -n "# Started log at: " >> rflog.txt
date >> rflog.txt
echo
echo "Log started, saved to $LOGFILE. Stop with Control-C"
while [ 1==1 ] ;
do
    wget -q 192.168.4.1/rfstat.txt -O - >> $LOGFILE
    # Adjust for desired sampling frequency (in seconds, floating point OK)
    sleep 1
```

```
done
# EOF


(Script delivered as file: ‘monitor_qcan2.sh’, live version is ‘monitor_qcan2_stdout.sh’)
```

Windows version: mon_qcan2.bat:

```
@echo off
rem #
rem # Script to monitor QCAN2 status, and create a logfile
rem # Written for Linux, ported to windows.
rem #
rem # Results are in rflog.txt
rem #

@echo *
@echo * Log started, saved to rflog.txt. Stop logging with Control-C
@echo *

:again
    rem # Get the latest table, append to file
    wget -q 192.168.4.1/rfstat.txt -O - >> rflog.txt
    rem  # Adjust for desired sampling frequency (in seconds, floating point OK on Linux)
    sleep 1
goto again

rem # EOF
```

Controller Command Flow

(This whole section is implemented, no new information is added / needed)

Intersection Controller Command Flow

(Implemented, this section is informational)

Included, a simplified table of intersection command flow. The rows are staggered, denoting the challenge and response structure of the communication. For full flow chart please see attached document titled: “Intersection Flowchart”.

Keys: (ZZ=zone) (VV=Vehicle) (00-FF=Hex ASCII characters) (OPT=optional)

BIT_7 0x80 = BLOCKED BIT_6 0x40 = IN_CONTROL
BIT_5 0x20 = ERROR BIT_4 0x10 = RESOLVING

Spaces between fields are added for readability.

AGV - QCAN2 Intersection Challenge / Response Table

AGV Sends	QCAN2 Responds	Status BITS	Condition/Comments
0x00 00 VV 00			Idle loop
	0x80 00 00 00		Idle loop response
0x01 ZZ VV 00			Anticipate Intersection (OPT)
	0x81 ZZ VV 00		Intersection is unoccupied
	0x81 ZZ 00 80	BIT_7	Intersection is occupied
	0x81 ZZ 00 A0/20	BIT_5 | ?	Radio Error
0x02 ZZ VV 00			Occupy Intersection
	0x82 ZZ VV 00		Intersection open
	0x82 ZZ VV FF	ALL	Intersection occupied (legacy)
	0x82 ZZ VV 10	BIT_4	Started resolving
	0x82 ZZ VV 90	BIT_7 | BIT_4	Preliminary NO
	0x82 ZZ VV 40	BIT_6	Intersection occupied by self

	0x82 ZZ VV 80	BIT_7	Intersection occupied by other
	0x82 ZZ VV C0	BIT_7 | BIT_6	Got Zone, but intersection occupied by other
	0x82 ZZ 00 20/A0/E0	BIT_5 | ?	Radio Error
0x00 ZZ VV 00			Release Intersection ZZ VV fields are optional
0x00 00 VV 00			Idle loop
	0x80 00 VV 00		Idle loop

Door Controller Command Flow

Included, a simplified table of door controller command flow. The rows are staggered, denoting the challenge and response structure of the communication.

Keys: (ZZ=zone) (VV=Vehicle) (00-FF=Hex ASCII characters) (OPT=optional) (SS=door status bits: Bit_0=input_1, Bit_1=input_2)

AGV QCAN2 Door Challenge / Response Table

AGV Sends	QCAN2 Responds	Comments
0x00 00 00 00		Idle loop
	0x80 00 00 00	Idle loop
0x03ZZVV00		Anticipate (Open) Door (OPT)
	0x83 ZZ VV 00	Door zone is unoccupied
	0x83 ZZ VV FF	Door zone is occupied (AGV Slow)
0x04 ZZ VV 00		Door Request
	0x81 ZZ VV 00	Door zone is unoccupied
	0x81 ZZ VV FF	Door zone is occupied (AGV Stop)

Open / Close Door	0x04 ZZ VV 81 – open 0x04 ZZ VV 82 – close	Open / Close Door
		The 'Read fixed equipment' command can be used to poll the door status
0x0a ZZ VV 00		Read Fixed Equipment
	0x81 ZZ VV SS	Door status bits relayed
0x00 ZZ VV 00		Release Door and door Zone ZZ VV optional
0x00 00 00 00		Idle loop
	0x80 00 00 00	Idle loop

Dispatch Command Flow.

Included, a simplified table of dispatch command flow. The rows are staggered, denoting the challenge and response structure of the communication. For full flow chart please see attached document titled: “Dispatch Flowchart” (under construction)

Terminology:

Dispatch QCAN2	D-QCAN2	The QCAN2 that is attached to the dispatch computer.
AGV QCAN2	A-QCAN2	The QCAN2 that is attached to the AGV

Table Keys: (ZZ=zone) (VV=Vehicle) (00-FF=Hex ASCII characters) (OPT=optional) (LL=Load Status: 01=Full FF=empty) (PP=Pick/Drop Command: 01=Pick, FF=Drop) (TT=Target / Destination Zone) (MM=Additional [More] Destinations: 00=Done, FF=More) (---- denotes state separation)

AGV QCAN2 Dispatch Challenge / Response Table

AGV Sends	A-QCAN2 Responds to AGV	A-QCAN2 Sends to D-QCAN2	D-QCAN2 Sends or Responds to A-QCAN2	Comments
0x00 00 00 00				
	0x80 00 00 00			
0x05 ZZ VV LL				Awaiting Dispatch
		0x05 ZZ VV LL		Dispatch QCAN2 builds a table of available AGVS
-----	----	----	----	----
			0x07 TT 00 00	Anticipate Vehicle
	Relayed Unmodified			
-----	----	----	----	----
			0x08 ZZ TT PP	Request Vehicle
		Relayed Unmodified		Dispatch QCAN2 responds from a table of available AGVS
	0x88 ZZ TT PP			
0x06 ZZ VV TT Accept Dispatch				
		Relayed Unmodified		
			0x09 ZZ VV TT	Confirm Vehicle
	0x89 ZZ VV TT			
				Dispatch Complete

0x06 ZZ VV FF Reject Dispatch				
		Relayed Unmodified		
			0x09 ZZ VV TT	Confirm Vehicle
	0x89 ZZ VV TT			
				Dispatch Complete

Fixed Equipment (DOOR) Relay Controls

Please Note:

This section is obsolete, the development thread was created on the assumption that door controller needs a switch based priority resolution

The formula for the door bit allocation is once a bit is occupied from one AGV, the other AGV(s) cannot reset it. Only after the first (and successive) AGV(s) release the door bit, it becomes possible to close it. This is to serve the intent, that once an AGV opened the door, it has exclusive control over releasing it.

Bit allocation table:

AGV_1 Door Bit 0	AGV_1 Door Bit 1	AGV_2 Door Bit 0	AGV_2 Door Bit 1	FE Relay_1	FE Relay_2
0	0	0	0	0	0
1	0	0	0	1	0
0	1	0	0	0	1
1	1	0	0	1	1

AGV_1 Door Bit 0	AGV_1 Door Bit 1	AGV_2 Door Bit 0	AGV_2 Door Bit 1	FE Relay_1	FE Relay_2
0	0	0	0	0	0
1	0	0	1	1	1
0	1	1	0	1	1
1	1	1	1	1	1

AGV_1 Door Bit 0	AGV_1 Door Bit 1	AGV_2 Door Bit 0	AGV_2 Door Bit 1	FE Relay_1	FE Relay_2
0	0	0	0	0	0
0	0	0	1	0	1
0	0	1	0	1	0

0	0	1	1	1	1

Please note that in this table AGV_1 and AGV_2 are interchangeable, so the third (and forth) table are redundant.

This bit table does not correspond to any standard (And / Or / Xor) table so we named it Central Resource Reset Protection table. (CRPT Table) Feedback needed / welcome.

Troubleshooting

The QCAN2 has many facilities to aid troubleshooting. Several distinct levels of diagnosis are available, from visual inspection of the LED s to looking at the serial communication to diagnosing the RF transmissions to connecting to the serial port.

1.) Visual inspection of LED-s.

Perhaps the most telling LED is the RF reception LED. It blinks when there is an RF packet received. This confirms that there is another QCAN2 in range.

The blinking serial LED confirms the presence of the RS232 signal. If the QCAN2 is connected to the AGV via the CAN bus, the corresponding CAN LED should blink.

The CPU LED confirms the operation of the CPU. It blinks in a constant manner, roughly one per second. (exact timing detailed elsewhere)

The three LEDs titled ‘STOP’ / ‘EVAL’ / ‘GO’ describe the current state of the AGV control. Using the traffic light metaphor, the red color LED is for STOP AGV, the green one is for GO AGV. The yellow colored ‘EVAL’ is the ‘evaluate’ LED has multiple functions, and it represents a transitional state. On Anticipate, the yellow stands for full speed AGV, the yellow and red stands for slow speed AGV.

2.) Command Line Diagnostics

This has been described extensively in the command line section.

3.) Hard Reset.

Holding the QCAN2’s top button for 12+seconds will reset the QCAN to its original state, like it was configured by the manufacturer.

Configuration error issues:

The QCAN2 Is resilient against configuration difficulties. Only two configuration cases are detrimental.

1. Duplicate AGV IDs

This is not fatal, as this is only used in duplicate occupier resolution. If more than one AGV has the same ID, the resolution falls back to the MAC address. (which is globally unique) The Duplicate AGV IDs can be solved by detecting them at the dispatch level.

2. More than one FE (Door) controller on the same zone.

When the QCAN2 powers up, it listens for the presence of a door controller on the same zone. If it detects a second door controller on the same zone, all the ‘traffic light’ LEDs flash, signaling zone conflict. (the ‘traffic light’ LEDs are the Red / Green / Yellow LEDs – Named STOP EVAL GO)

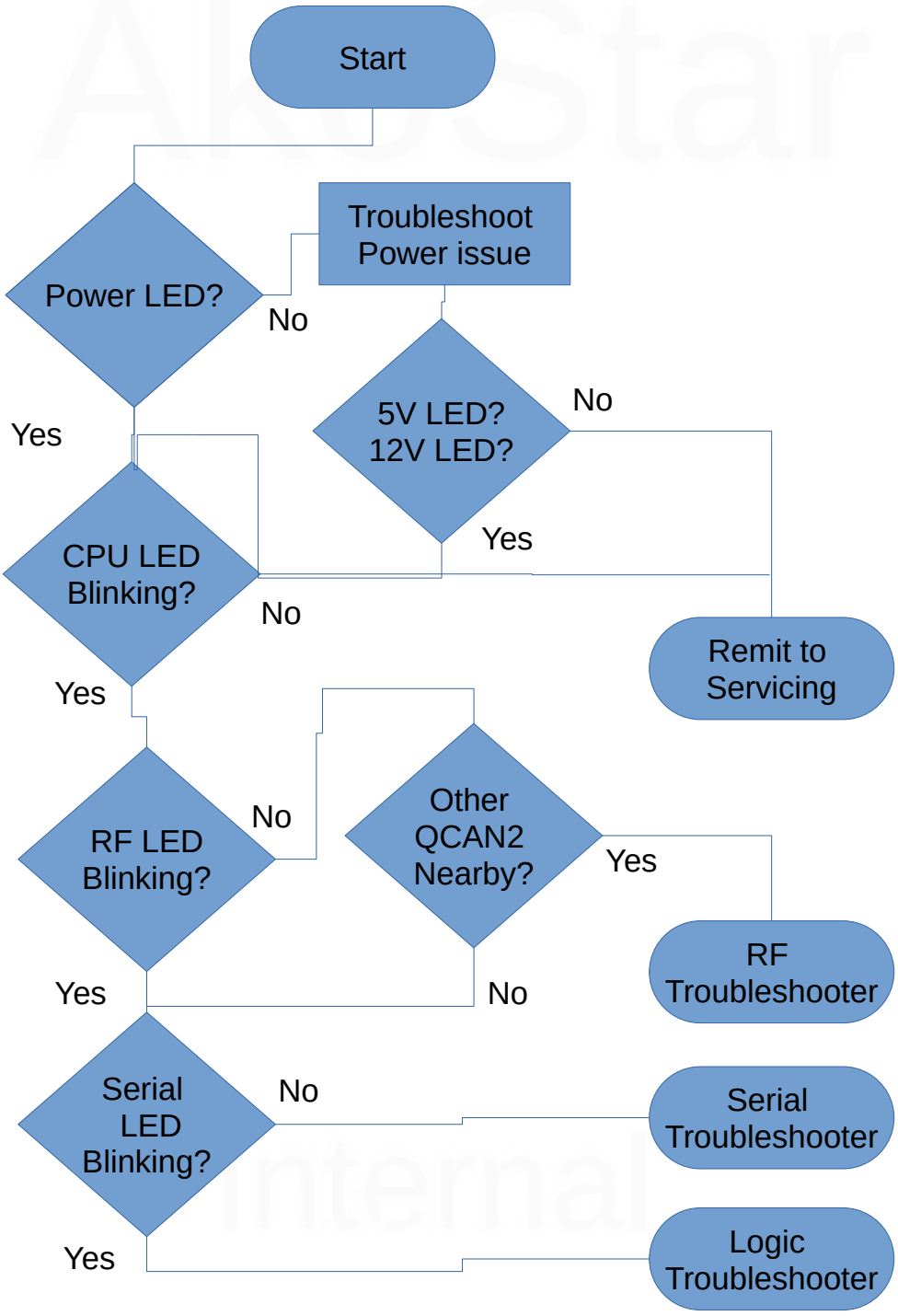
The QCAN2 detects this condition on boot up, as the second FE can detect if there is already a FE within the same zone. This way the error could be signaled early, to prevent the two controllers from mixing up communication and interfering with one another. The two FEs on the same zone can deliver conflicting information, the symptom of which is intermittent switch state indications.

Diagnostic Charts:

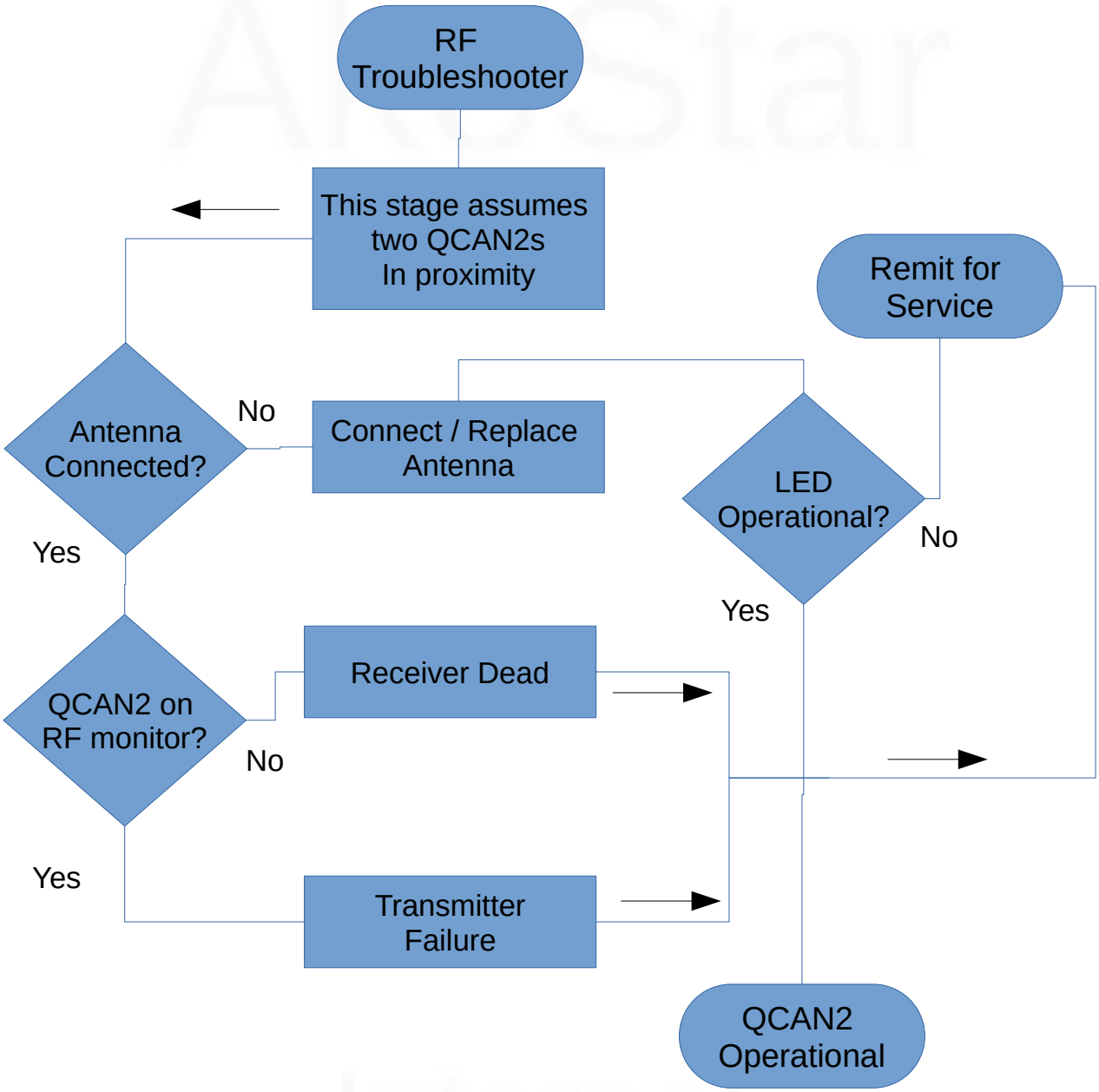
The following charts can be used to run the diagnoses.

This part of the page is intentionally left blank.

General Troubleshooter:

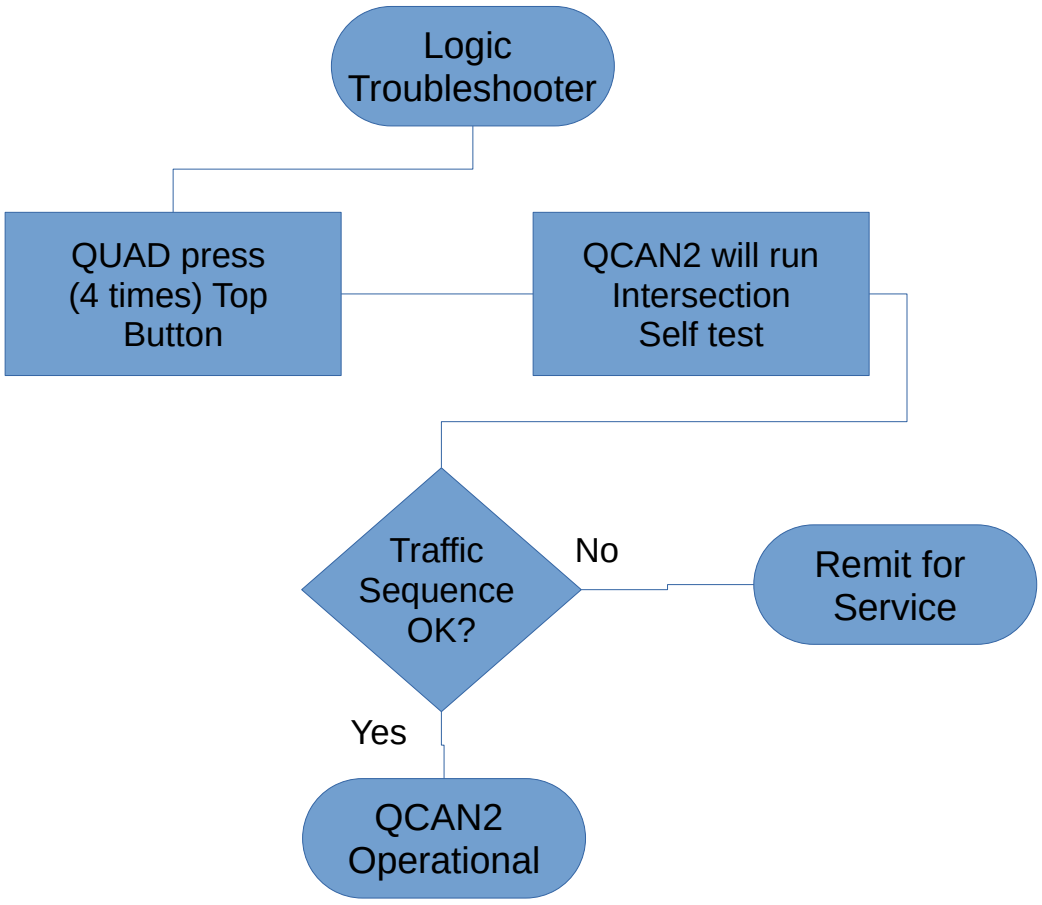


RF Troubleshooter:



Logic Troubleshooter

The QCAN2 has an self test function built in. After activating the self test, the QCAN2 cycles between intersection states in 5 sec to 15 sec interval. The self test zone is 100 (decimal). By activating the self test on more than one QCAN2, it is observable that they obey each other's occupy stages. To activate the self test, press the QCAN2 button four times in succession. (Like a mouse double click, but four clicks) The LEDs indicate as the device cycles.



Supplementary Tools

The QCAN2 system is extremely versatile. It has an additional USB port that can be connected to a terminal emulator. For example, a tablet PC with **Putty installed. With this setup, the connected QCAN2 can monitor the status of any other QCAN2's, or it can issue commands to any other QCAN2.

The terminal is interacting with a shell-like command interpreter, where various commands can be issued. There are commands to inspect the RF table, the State table, optionally Start and Stop the AGV, emulate any event coming from the Intersection / Door controller, or the dispatcher. This command interpreter can also be used as a configuration tool, a testing and troubleshooting tool. An application is under construction to permit interaction with this command interpreter via simple button presses. For more information see QCAN2 command interpreter documentation.

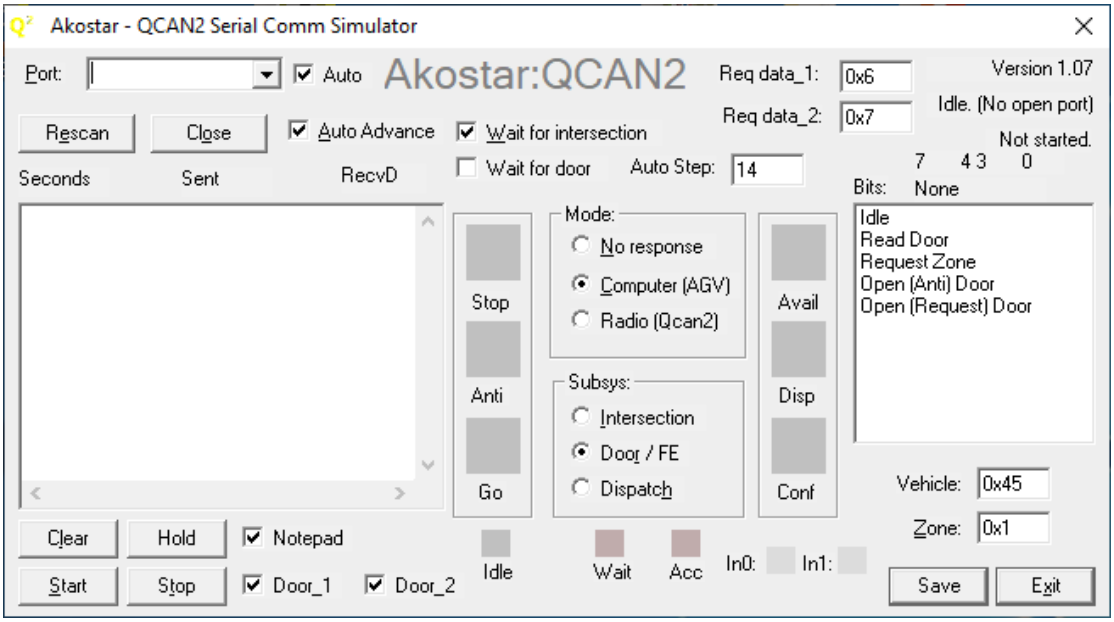
** (Putty: Open Source terminal program)

Simulation

(This section is informational)

Windows Based Simulation

This simulator has matching functionality to the Savant QcanComm program. Technically it is qualified as a controller. The Akostar Simulator / Controller has some additional functionality to aid automated testing. (screenshot may differ from image due to updates)



The serial comm simulator was developed to automate testing, and expand the range of parameters that can be modified. All text boxes are ‘live’ controls, the typed values take effect immediately. However, most of the QCAN2 commands operate on a ‘delta’ (that is, it is sensitive to command change), so some ‘live’ parameters are only acted upon if the command has changed. This is especially visible on the dispatch command, where the except / reject dispatch takes effect after command change. In other words, to emulate the command change on request vehicle, one needs to switch back to anticipate and than change the parameter. Switching back to request, the new parameter takes effect.

The Auto Advance system: the states of the QCAN’s communication are advanced automatically. When the states are at the end state, the auto advance wraps around. The states are stepped forward at the ‘Auto Step Timer’ interval, which assumes a random value at startup. The ‘Wait intersection’ feature stops the auto-advance from switching until the AGV acquires the resource.

Description of controls (see screenshot for details):

Auto Advance ON	Alt-A	Enables Auto Advance
Auto Step Timer (sec)		Specifies the interval between auto step stages
Start	Alt-S	Starts the Communication
Stop	Alt-T	Stops communcation
Mode		Mode of operation
Subsystem		The QCAN sub systems
Vehicle		Hex number of the current vehicle
Zone	Alt-Z	Hex number of the current zone
Serial Port	Alt-P	Select serial port
Rescan	Alt-E	Re-scans serial ports
Close	Alt-O	Closes (free-s) current serial port
Wait	Alt-W	Wait for intersection availability on auto step
Dump		Save comm details (Not implemented)
Clear	Alt-L	Clear Comm listbox
Toggle Hold	Alt-H	Toggle hold on listbox, hold lets one examine contents
Add random chars	Alt-R	Taint communication with random characters at random places
Exit	Alt-X	Close program

The standard windows keys will operate as expected (like: Alt-F4 to close program)

Simplified operating procedure:

1. Select COM port
2. Click on desired mode (Computer)
3. Click on subsystem (Intersection)
4. Select desired Command (Idle)
5. Click on Start

The selections can be executed in any order.

Updates to the Simulator(s):

Back-porting the QCAN2 python simulator for Windows.

Updated text boxes / numeric entries, they now accept both decimal and hexadecimal numbers. Hexadecimal numbers have the 0x prefix. (0x20 = 32)

- Bit field display for easy reading of status_2 bits. (reading set / reset states)
- Save file as 4 digit name templates → wqcan2_0000.txt
- The simulation sends real time entries to the program notepad.exe
- Shrunk the GUI, so four QCAN2s fits on one screen.
- Added fields for data_1 and data_2 for dispatch requests. The field values are transmitted on Dispatch → Request Vehicle.
- Saved state of most every action, the simulation attempts to restart in its previous state
- The simulation auto connects to the next available port (if ‘Auto’ checkbox next to port is checked)

To activate the notepad feature, start the windows notepad program, and in the simulation click on the check box titled ‘Notepad’. The simulator will broadcast the event string to the running notepad program. The lines are prefixed with the name of the serial port, the transaction serial number and the (semi) real time of the entry to be displayed.

Once the data is in notepad, one is free edit / save / delete. The notepad feature does not have any limits testing, and notepad will easily take several tens of megabytes of data. All the other features have limits testing, they can be run indefinitely, supporting regression test cases.

Please note, that on testing the dispatch functions, that the system operates on command delta (change of command). So if one wants to test a different parameter, it has to be via transition to / from a different command. (implementing parameter delta would be contrary to current theory of operations)

For example:

wait dispatch to -> accept dispatch positive (green)

or

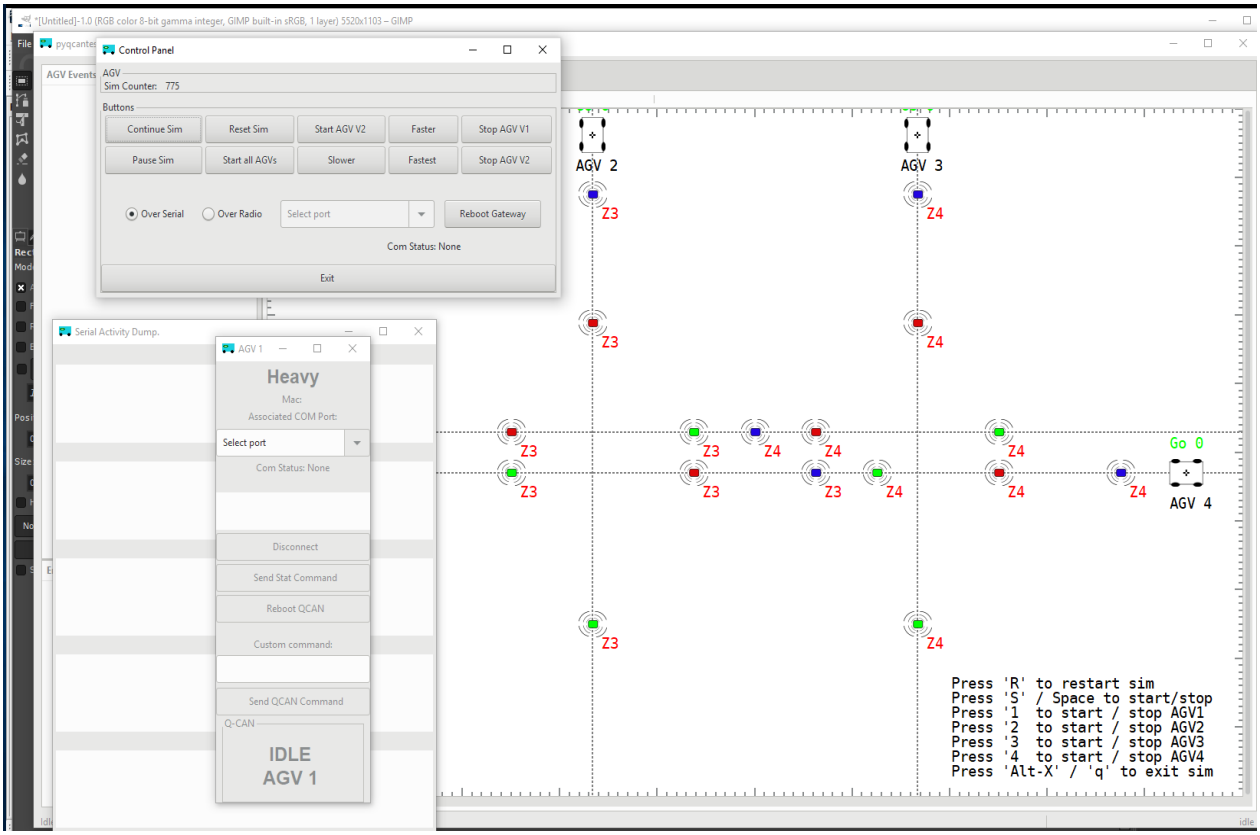
wait dispatch to -> accept dispatch negative (red)

one can switch back and forth between wait → accept pos → wait → accept neg

Python Based Simulators

We have also added a powerful simulation software, that sends commands to the QCAN2 serial port, much like the AGV would. The simulation then observes the QCAN2 responses. The responses are then interpreted, and an on-screen AGV simulates the actions of the physical vehicle. The simulation accurately models the requirements of the AGV control. It has been an instrumental tool to create a protocol and set of state machine states that are immune to RF disturbances, and other anomalies.

On the screenshot below, five AGVs travel a pre-drawn path. At startup, they all assume a random speed (within range), and they communicate with their respective QCAN2s via real RS-232. The QCAN2s communicate with each other over RF, and respond to the AGVs requests. The simulated AGVs behave like the real AGV, obeying the STOP/SLOW/GO commands. The simulation below depicts five AGVs coordinate over 2 zones. This simulation has helped us overcome many of the challenges associated with development.



This simulation has yielded us considerable insight onto the challenges we face. We (re) discovered the phenomena of the duplicate bully, could see the delays in communication and response, and could tune the QCAN2 state machine to adapt to the challenges of the real world RF uncertainty. It is also included and installed in the provided laptop.

AGV Visual Simulation Setup

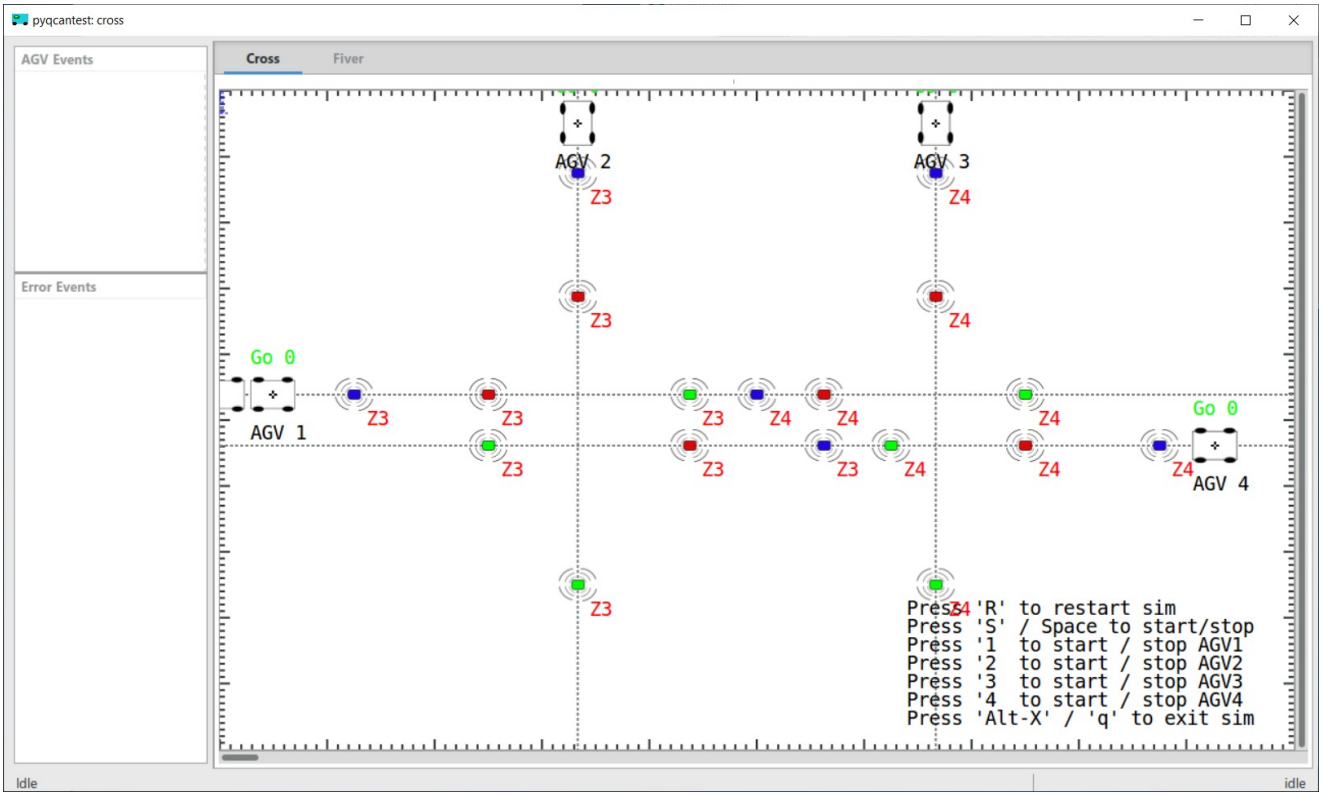
The AGV Visual Simulation is a useful tool to verify and demonstrate AGV intersection interactions. We installed it on the laptop, so it is ready to go with a click of a button.

Setup process:

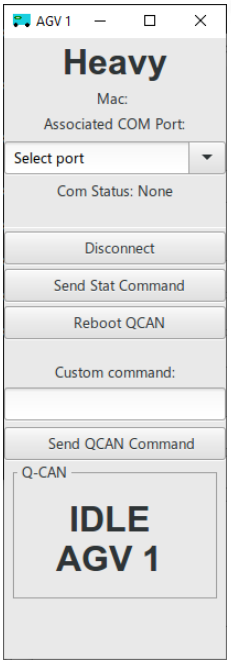
- Connect QCAN2s USB ports to the PC
- Start simulation (shortcut icon on desktop called ‘Visual Sim’)
- Visually connect the AGVs to serial ports (see Serial Setup Screen Shot)
- On the Control Panel of the Simulator, Click the ‘Start all AGVs’ button.

One may use the Alt-Tab key to navigate between the Visual Simulator’s windows. Alt-X key will set focus to the main screen, and if foxus is already on the main screen, Alt-X will exit the Visual Simulator.

Visual Simulation



The Visual Simulation was meant for a multi-screen desktop computer, where all the controls, configurations and views can be shown at once. However, all the functionality can be exercised from a single screen. Use Alt-Tab to pan between them There is a tab control on top of the screen where a different scenario is pre-loaded.



The Serial Screen

Setting up the Visual Simulator is relatively simple. All one has to do is assign serial ports to AGVs. The screen short on the left is the Serial Setup window, depicts the ‘Select port’ item. Pull the combo box’s selection list down, and select a serial port from the listed items.

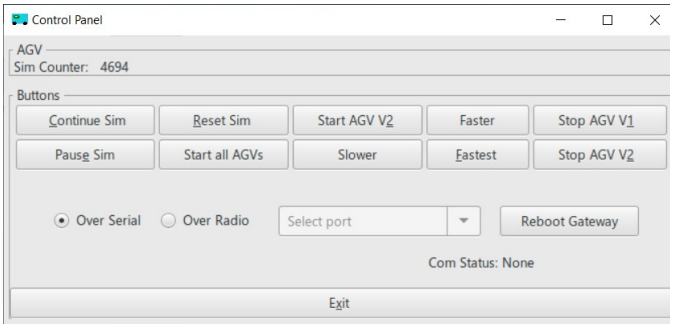
The AGVs can have simulation names, this particular one is called ‘Heavy’, because it is the double AGV on screen. The names have no influence of the workings of the Visual Simulator.

This window can also be used as a control window for the QCAN2, issuing reset, and send any command via the USB serial port.

Starting / Stopping the Simulation / AGVs

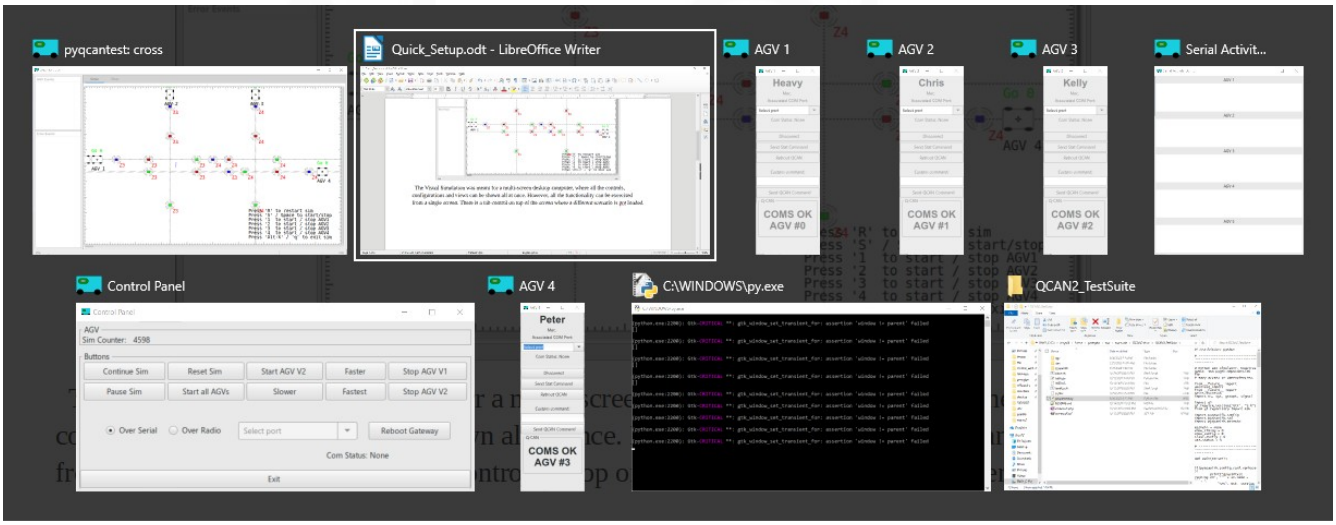
The simulator can be controlled from this panel. There are buttons to Start / Stop the simulation, to restart / reset the simulation, speed things up, slow things down.

This panel represents the evolution of the Visual Simulator where we tested different aspects of the system, they where relevant for development.



There are several more windows in the Visual Simulator, most of which was needed during development. One of the panels show serial communication between the Visual Simulator and the QCAN2. While all this was essential during development, it is safe to ignore for demonstration purposes.

The following screen shot depicts all the process windows on my current system. Please note the windows that belong to the Visual Simulation.



Dependencies:

The dependencies of the Visual simulation are all open source components. They are in use by a wide array of audiences, including large corporations. Here is a short list for Windows:

1. python (any version);
2. pygi;
3. pygobject;
4. pyserial;

The dependencies are installed on the laptop, and included in the QCAN2 directory on the desktop.

Multi Platform:

The simulation works on every platform that has python / pygi. This includes:

- Windows (All versions after Vista)
- Linux (All major distributions) – Tested on Ubuntu and Fedora
- Chrome Book – on older versions one can side install Linux, newer versions have Linux support built in
- Mac – Untested, but given that all other platforms work, it usually works well

Tool Summary:

The Visual Simulation was a great tool for testing during development, and it may serve as a powerful addition for Quality Control and Demonstration. There are versions for the door subsystem, and versions that drive the internals of the QCAN2 in more detail. New version for the Dispatch exists.

The AirMon utility

We have created several new diagnostic / testing tools for the QCAN2 product line. Also ported some of the existing tools to Linux based utilities using the Python / PyGObject framework. The rationale behind it was to create a unified Embedded / Development / Testing platform. The advantage is cleaner end results and higher productivity.

This particular utility shows all AGV activities within RF range. It uses the familiar traffic light metaphor to represent individual AGV statuses. Red for stop, Green for go, Yellow for anticipate, Yellow/Red for anticipate in slow mode. The utility can filter by zone, so a single intersection in a larger installation can be monitored. The door (FE) activity may be monitored the same way, but currently there are no arrangements made to show switch and relay statuses.

AirMon Setup

The setup is simple, just connect and QCAN2 to a PCs USB port. Connect the AirMon Serial port to the QCAN via the drop down selection, and press … start.

The QCAN2s USB port acts as a command line terminal, and this utility programmatic-ally queries the RF table. It then interprets this information, and parses the result in an easy to read format, including a traffic light like display.

Up to eight AGV statuses are displayed, but there is no limit to the controls one can put on screen. The additional feature of filtering the activity by zone allows even large installations to monitor AGV activity. The UI code and query code can be easily separated, which allows the AirMon utility to be ported to any other Platform / Language.

Installing PyGObject on Windows.

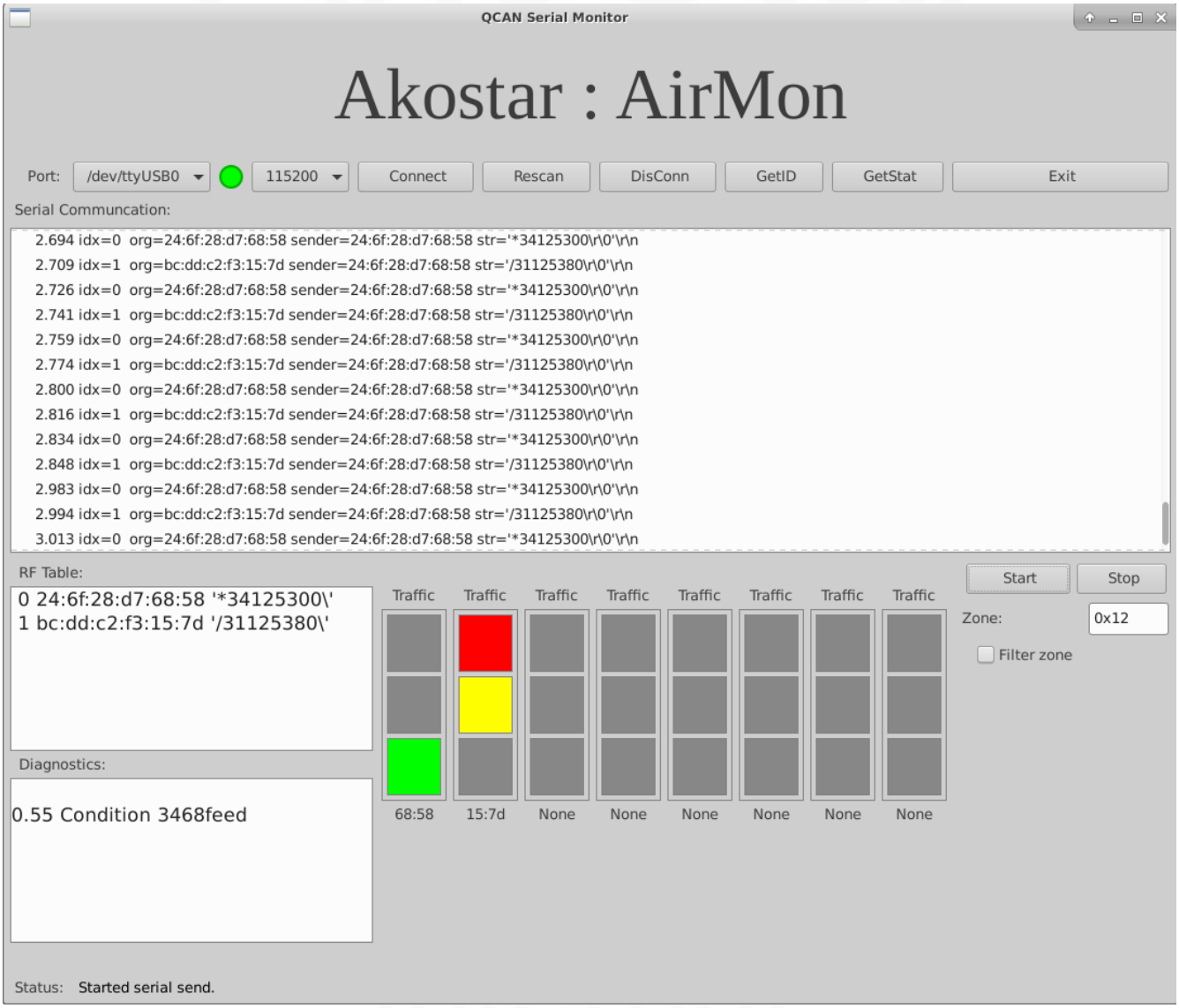
To run most of these utilities, one needs to install the PyGobject dependencies. Installation is trivial, two parts: python 2.7 and PyGObject.

Following, are the download links to the install ready resources.

https://www.python.org/downloads/release/python-2716/
https://sourceforge.net/projects/pygobjectwin32/

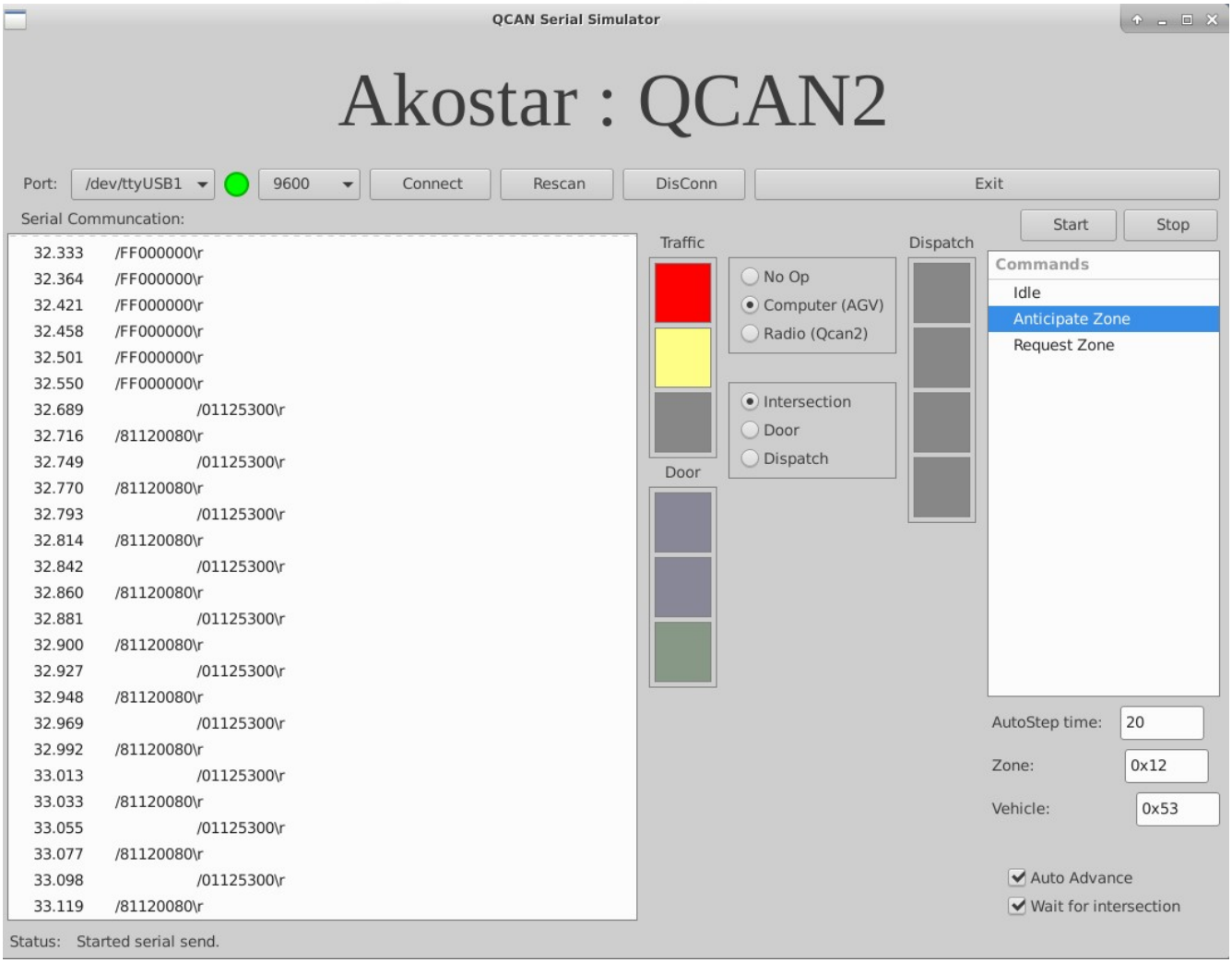
These files are also included and installed on the demo laptop to represent a proof of concept.

The advantage of the above toolkit is that it runs on Windows / Linux / Mac / Chrome Book with little or no modification. The added effort to install the PyGObject framework is negligible compared to the productivity gain and feature additions made possible by it. The PyGObject framework exist for Windows 10 as well.

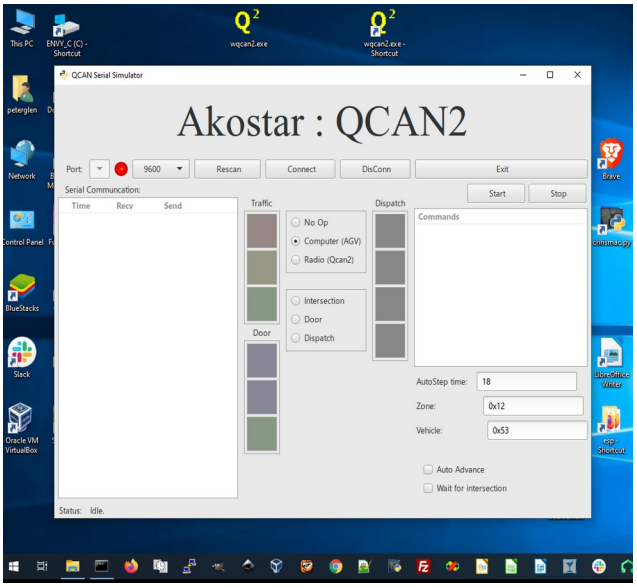


Python QCAN2 Simulator / Control:

This is the familiar simulation software crafted in Linux / Python. Because of the higher level language, we added some new features; more tests, and more detailed reporting. The dispatch code is not included yet; will be completed in the next step.



Python QCAN2 Simulation Running on Windows:



To the left is a screen shot of the simulation utility running on windows:

It has identical functionality the Linux counterpart, and a similar appearance.

Note: this utility has been partially updated to drive the dispatch subsystem, as per request, we switched back to the windows ‘C’ based utility.

GUI based configuration tool.

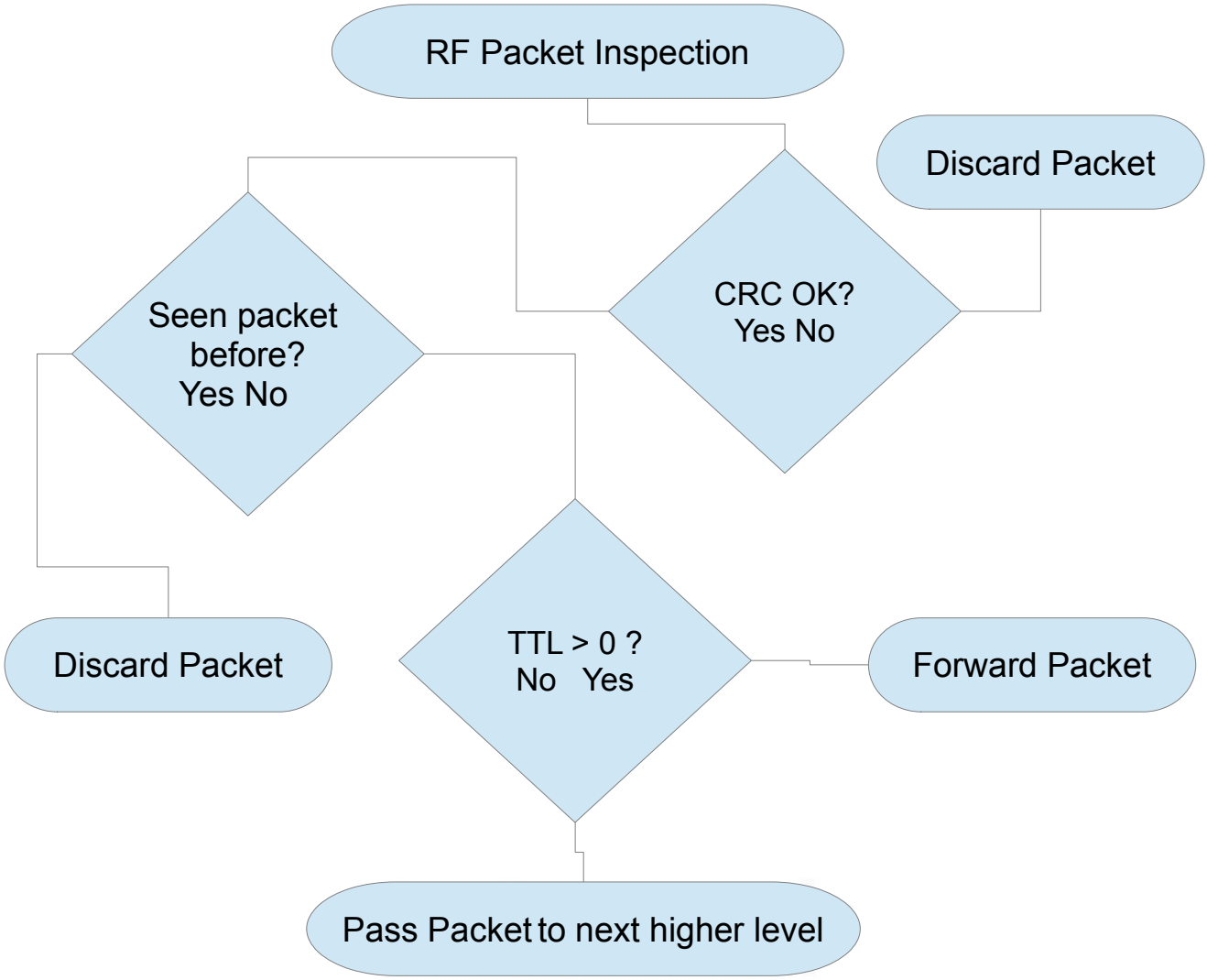
While this is not yet in the works, we envision a Graphical window with zone number fields, door-zone number fields, auto-start flag fields. This GUI window can be used to program / configure the QCAN2 via the USB port. The programming then can be automated on a per customer basis or read from a database.

TODO.

Algorithms Used in the QCAN2

There are many algorithms that make up the QCAN2 protocol. The following chart depict the hierarchy of the algorithms used.

On the top of the stack there is the RF subsystem.



Future development:

The QCAN2 was implemented to legacy specifications. Technology has moved forward since the original specification, new tools, faster processors, modern RF and high speed communications have become available. This gives rise to a modernization opportunity to the QCAN2.

In the following chapters we outline a specification proposal, based upon the learning acquired during the development of the QCAN2 device. It is not an error list, or defect list, it is a guide wire for a possible future direction in implementing QCAN3.

The QCAN3 advancement

The QCAN3 may move towards more real time interaction. The old device had a 400-500 ms serial transmission cycle, and the data was encapsulated within that packet. This was done with the intention of uniting the ‘stay alive’ and the ‘communication protocol’ messages.

The new device would separate them into two distinctive streams: a.) stay alive and b.) data;

The stay alive could be shorter, for example 4 bytes. (CAN bus native length)

The data could be longer, for example 8 bytes; (2 x CAN bus native length)

The rational for the separation, is that the stay alive can function independently from the main data stream. This way the main data stream can be event driven; which allows faster response times. Also delivered content can be verified by an immediate return transmission.

For instance the ‘stay alive could be delivered every 500 ms, 1000 ms being cause for timeout response. (2 missed packets)

The data packet would contain information similar than the legacy devices, but the extra bytes would allow for a more consistent information flow. An example would be an additional op code for packet type identification. An op code for error conditions. A return packet could contain information about the reception / acknowledgment of the data, or the status of the operation itself.

Avoiding the cementing of the details, here is a pseudo code that would achieve that:

PP=preable TT=type_code CC=command_code P1=param_1 P2=param_2 P3=param3 CS=checksum

PP TT CC P1 P2 P3 CS

And the matching return code:

PP TT CC R1 R2 R3 CS

Assuming the preamble of 0xab, type code 0x01, here is how an occupy (02) zone 0x03 by AGV 0x45 would look like:

ab 01 02 03 45 00 00 CS

The checksum could be any algorithm agreed to by both AGV and QCAN2; for example:

```
uint16_t sum_packet(const uint8_t *ptr, int len)
{
      uint16_t sum = 0;          // Allow overflow
      for(loop = 0; loop < len; loop++)
            sum += ptr[aa];      // Per byte
      return sum;
}
```

The sum is checked by both peers, bad packets are discarded. The event is interpreted immediately, submitted to the QCAN3 state machine, and sent back as soon as data is available. This is different than the legacy device, it delivered information on the next round robin of the serial cycle.

The RF can also be adapted to follow the semi real time model. Instead of following the round robin cycle of the incoming commands, one can deliver an immediate update to the RF subsystem, and accordingly, receive an immediate update.

Events initialed by the QCAN3. The AGV is listening to async communications from the QCAN3 as well. External events like intersection events; door open events may reach the QCAN3 immediately.

Centralized Intersection controller.

The peer to peer intersection protocol theory has an inherent flow. The control is established peer to peer, relying on the participants to obey a controller peer. However, if the communication to the controller entity is severed, the other peers interpret it as a go signal. The resource relied on positive acknowledgment for negative action.

The other theory is based upon the theory of a dedicated intersection controller. The resource is now controlled by a central resource controller, turning the cycle of positive acknowledgment to a positive action. If the communication to the central resource controller is severed, it is interpreted as a negative action, stopping the AGV. This assures that the AGV only goes if it has permission from the central resource controller, which results in a failure free theory of intersection controller. This theory can also be implemented in the QCAN2. The optimization is fail free theory vs. the additional cost of QCANX controllers.

Summary:

The current state of QCAN2 is near completion. The intersection logic is extensively tested, the door controllers work as expected, and the dispatch mechanism is functioning. The dispatch mechanism may need review, especially in corner cases.

We welcome your feedback;

Chris Bogue / Peter Glen