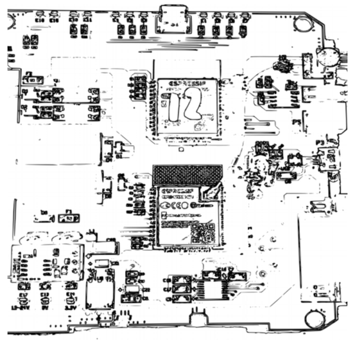
QCAN2

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

Developer’s Manual

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-14-2020	Peter Glen	Preparation for prototype delivery	
2.0	Jan-15-2020	Peter Glen / Chris Bogue	Review	
3.0	Mar-24-2020	Peter Glen	Doc release	
3.0	Mar-25-2020	Peter Glen / Chris Bogue	Review	

This part of the page is left intentionally blank

Table of Contents

This part of the page is left intentionally blank

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.

QCAN2 is the successor of QCAN, implemented with modern processors, and powerful, updated RF technology. The device maintains connector and serial port compatibility, and similar command response as it 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 QCAN2 we managed to resolve most - if not all - of the challenges of its predecessor. 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 device 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 a two powerful simulation software suites. One that sends commands to the QCAN2 serial port. The simulation then observes the QCAN2 responses. The other is a simulation is a Visual Modeling program, that simulates AGV action on screen. The on screen 'virtual AGVs' obey the instructions of 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.

This document also contains legacy descriptions. However, some 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.

In the following chapters we will briefly introduce each and every mode of operation.

Current State of Implementation

This is a partial list of ‘git log’. Items marked ‘autocheck’ are initiated from the Makefile.

Date:	Tue Mar 24 17:44:39 2020 -0400	autocheck
Date:	Tue Mar 24 16:36:32 2020 -0400	usb_exports
Date:	Tue Mar 24 16:33:48 2020 -0400	subproj
Date:	Tue Mar 24 16:29:36 2020 -0400	imported_windows_stuff
Date:	Mon Jan 6 12:25:39 2020 -0500	Can data replication and monitor
Date:	Tue Dec 24 19:21:37 2019 -0500	Christmas_edition
Date:	Fri Sep 6 16:26:06 2019 -0400	Win test prog complete, QCAN2 whatsup speed update
Date:	Thu Sep 5 18:53:29 2019 -0400	Added develop subdir
Date:	Fri Mar 22 22:28:35 2019 -0400	Inter processor comm
Date:	Sun Mar 17 21:32:32 2019 -0400	UI_added
Date:	Sat Mar 16 15:27:52 2019 -0400	Recomp on linux
Date:	Wed Jan 16 23:40:05 2019 -0500	CAN bus works
Date:	Tue Jan 15 17:31:18 2019 -0500	Added_docs
Date:	Fri Jan 11 12:43:01 2019 -0500	Connect_sim
Date:	Wed Jan 9 17:35:24 2019 -0500	Benset system
Date:	Fri Jan 4 18:05:35 2019 -0500	Started_sim_command_via_rf
Date:	Tue Jan 1 18:21:30 2019 -0500	Intersection crossing test passes
Date:	Mon Dec 31 18:00:35 2018 -0500	Multi Intersection Logic

Milestones:

(25-mar-2020) Revision 1.07 Released
(5-mar-2020) Revision 1.03 Released
(25-feb-2020) Revision 1.02 Released
(22-jan-2020) Getting ready for prototype delivery. Every subsystem is implemented,
(dec-2019) The intersection peer to peer-to-peer logic and the simulation harness is implemented. The radio frequency infrastructure is in place, state tables and command information are reliably passed between QCAN2s.

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		Offers Traceability
Simplified Setup	Jumpers	None	Zero Configuration

This part of the page is left intentionally blank

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

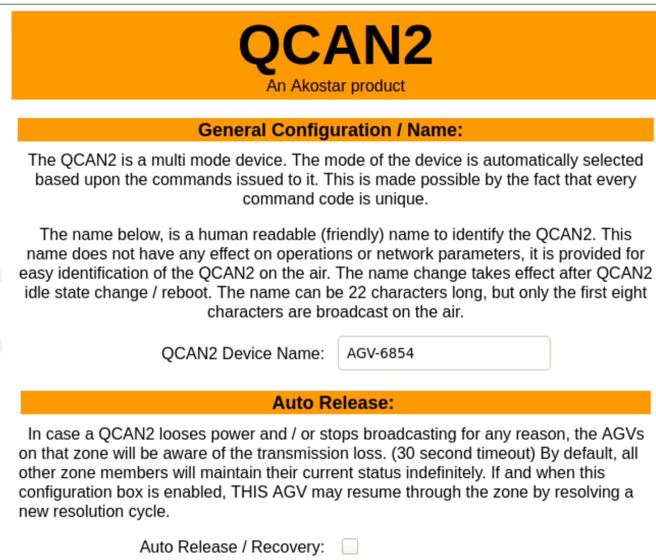
Configuring the QCAN2 via the web interface is extremely convenient. The QCAN2 will expose itself on the WiFi name space as "QCAN2-NNNN" (without the quotes), where NNNN is the last four digits of the QCAN2's MAC address. The WiFi password is pre-set to '12345678' (no quotes) The QCAN2 will listen on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

Configuration main page

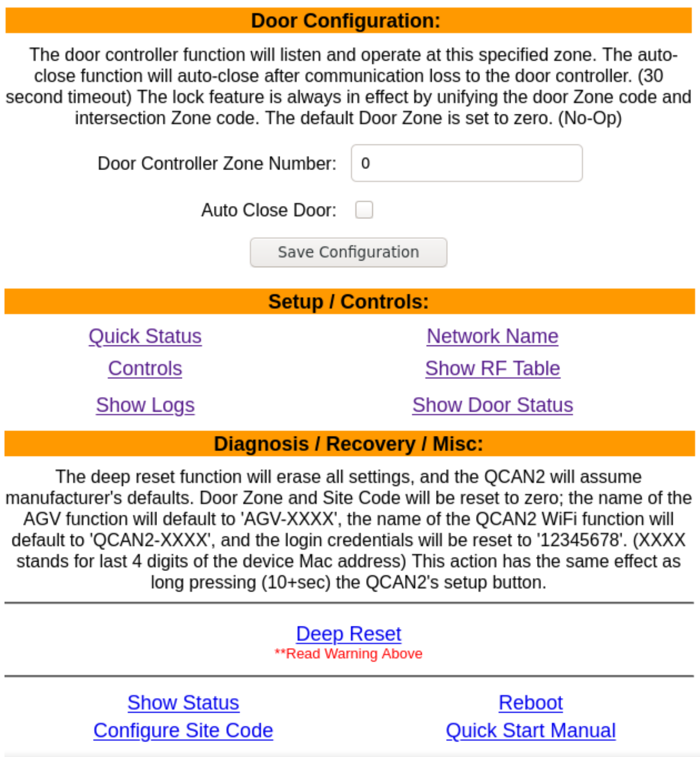
To the right, is a screen shot of the initial page the QCAN2 displays when connected to.

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:

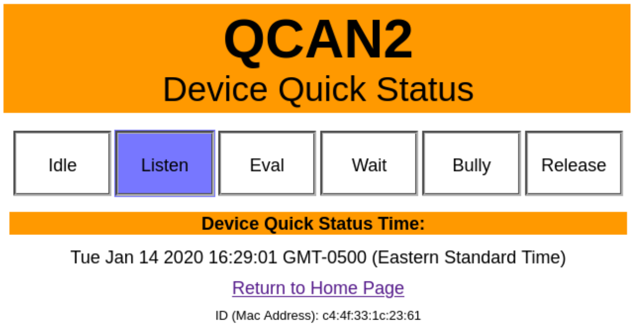
Door Zone. This is the zone the door controller listens to. The default is zero, which means no door controller function is active.

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.

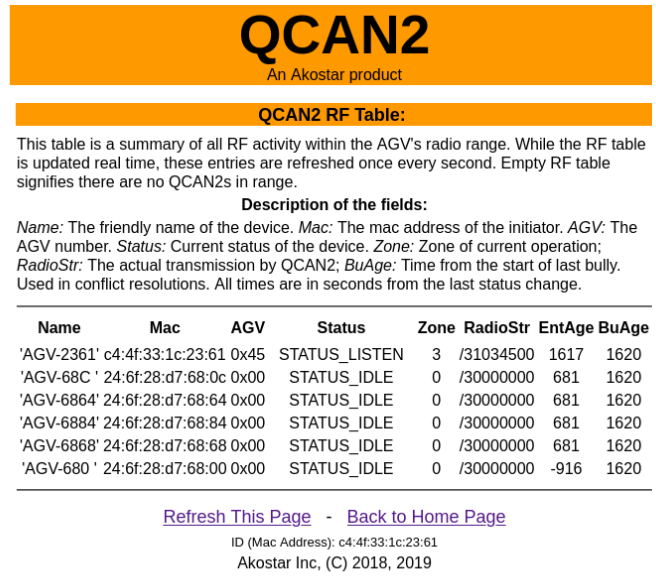
Informational Items



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.



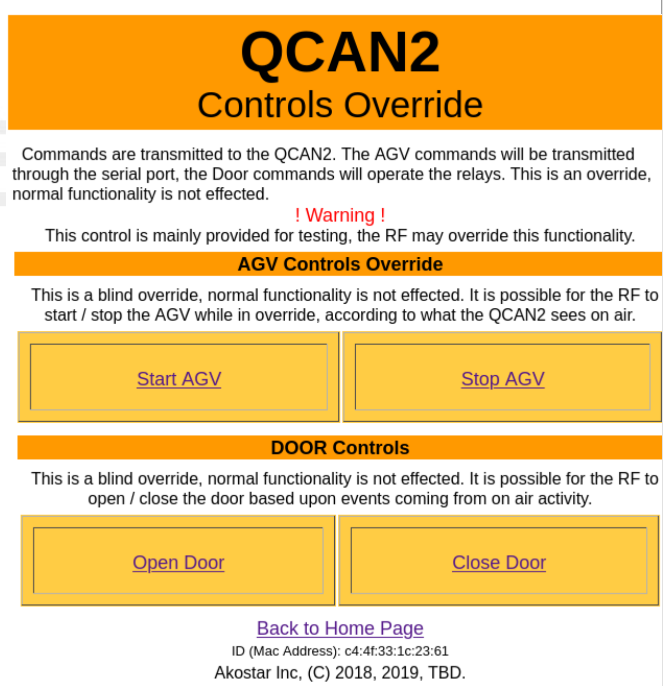
Name	Mac	AGV	Status	Zone	RadioStr	EntAge	BuAge
'AGV-2361'	c4:4f:33:1c:23:61	0x45	STATUS_LISTEN	3	/31034500	1617	1620
'AGV-68C '	24:6f:28:d7:68:0c	0x00	STATUS_IDLE	0	/30000000	681	1620
'AGV-6864'	24:6f:28:d7:68:64	0x00	STATUS_IDLE	0	/30000000	681	1620
'AGV-6884'	24:6f:28:d7:68:84	0x00	STATUS_IDLE	0	/30000000	681	1620
'AGV-6868'	24:6f:28:d7:68:68	0x00	STATUS_IDLE	0	/30000000	681	1620
'AGV-680 '	24:6f:28:d7:68:00	0x00	STATUS_IDLE	0	/30000000	-916	1620

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.



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 will reset the Manufacturer's configuration. (pass to: 12345678)

WiFi Device Compatibility

The Configuration Items operate in all common platforms and browsers. We tested PC / Apple / Unix / Android devices, all of which showed complete compatibility. We tested Firefox / Chrome / Safari / Edge, and it showed seemingly 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 action 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. 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 service calls, one can install a new QCAN2 without any tools or configuration.*

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, 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.

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. 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.

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 configuration 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 item in 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 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.

This part of the page is left intentionally blank

Command Interpreter

The QCAN debug 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 status. The terminal connection is available via the USB connector next to the serial port.

The command line

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

The serial port is then allocated by Windows, for example it sees COM6. One can confirm which port with 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)

Typing help at the command prompt will deliver a list of commands. Typing help <command> will deliver a short intro on the command. For the door zone command you may use the command: ‘zone’ (no quotes) To set the door zone to 5, use: ‘zone 5’

Commands

The following commands (and more) are recognized:

"macs" "leak" "can" "xcan" "xrfmon" "stat" "check" "ls" "dump" "bdump" "set" "get"
"stop" "start" "?" "help" "clear" "echo" "noop" "logdel" "log" "cpu" "nvs" "mem"
"m" "reboot" "aclose" "list" "ant" "req" "rel" "auto" "id" "name" "zone" "azone"
"mode" "verbose" "force" "bench" "benset" "pok" "rfmon" "hello" "serial"
"version" "ver" "stale" "switch"

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

Command Short List

The commands below we used extensively, so it is described here in detail. For more information see the document titled comline.odt.

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.
stat	Display current status of QCAN2 state machine, MAC address, and auto status.
ls	List RF table entries
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 normal operation.

An example command line:

QCAN>> zone

This above command will print the 'self' door zone number of the AGV.

The interpreter is connected to a dumb terminal. (via USB Serial Converter) It has minimal editing key facilities, but the Backspace key operates as expected. (No Arrow Keys) In case of an erroneous entry, press the Enter key to start a fresh command line. QCAN2 will re-issue its prompt: "QCAN2 >>"

Examples:

The output of the stat (short for status) command listed below:

QCAN2>> stat

```
QCAN2 version 1.07
States:
  Mainstate: QCAN_STATUS_IDLE zone=0 par1=0x0 par2=0x0
  Doorstate: QCAN_STATUS_IDLE zone=0 par1=0x0 par2=0x0
  Dispstate: QCAN_STATUS_IDLE zone=0 par1=0x0 par2=0x0
Configs:
  Forwarder: 0   Repeater: 0
  Autoclose: 1  Autorelease: 1
  Self Door Zone: 2
  Legacy Mode: 0
Properties:
  Mac: 24:6f:28:d7:68:58
  AGV Name: AGV-6858
  Last Mon: 444
  Boot count: 31
Diagnostics:
  rfmon: 0  xrfmon: 0
  forwmon: 0  xforwmon: 0
  canmon: 0  xcanmon: 0  sermon: 0
  Auto step status: 0  Leak test: 0
OK
```

RF Safety, Coexistence

The QCAN2 has two radios. One for configuration, one for RF negotiation, the main radio. The RF 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)

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.

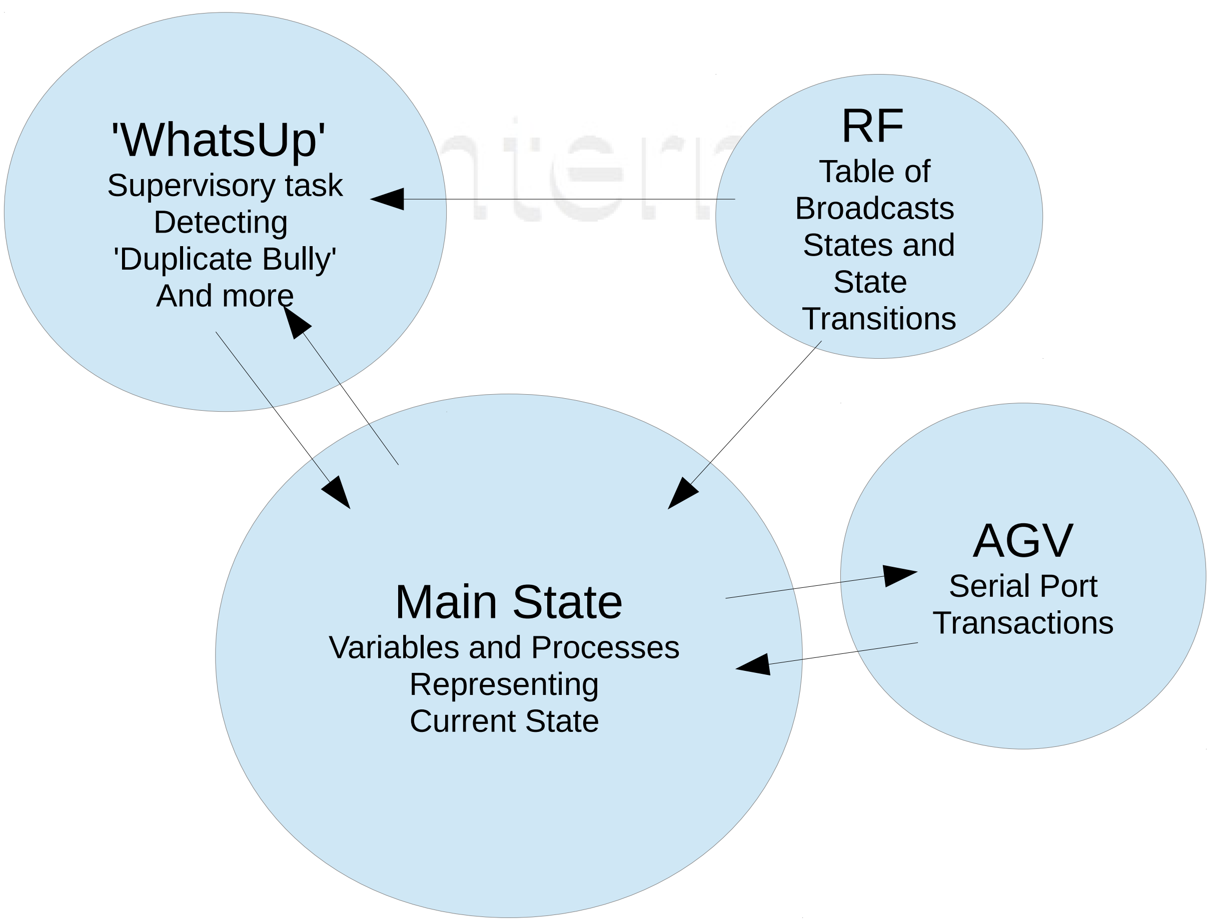
This part of the page is left intentionally blank

State diagrams

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

QCAN_STATUS_IDLE
QCAN_STATUS_LISTEN
QCAN_STATUS_EVAL
QCAN_STATUS_WAIT
QCAN_STATUS_BULLY
QCAN_STATUS_RELEASE

QCAN_STATUS_DOOR_OPEN
QCAN_STATUS_DOOR_WAIT
QCAN_STATUS_DOOR_OPENED
QCAN_STATUS_DOOR_GO



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.

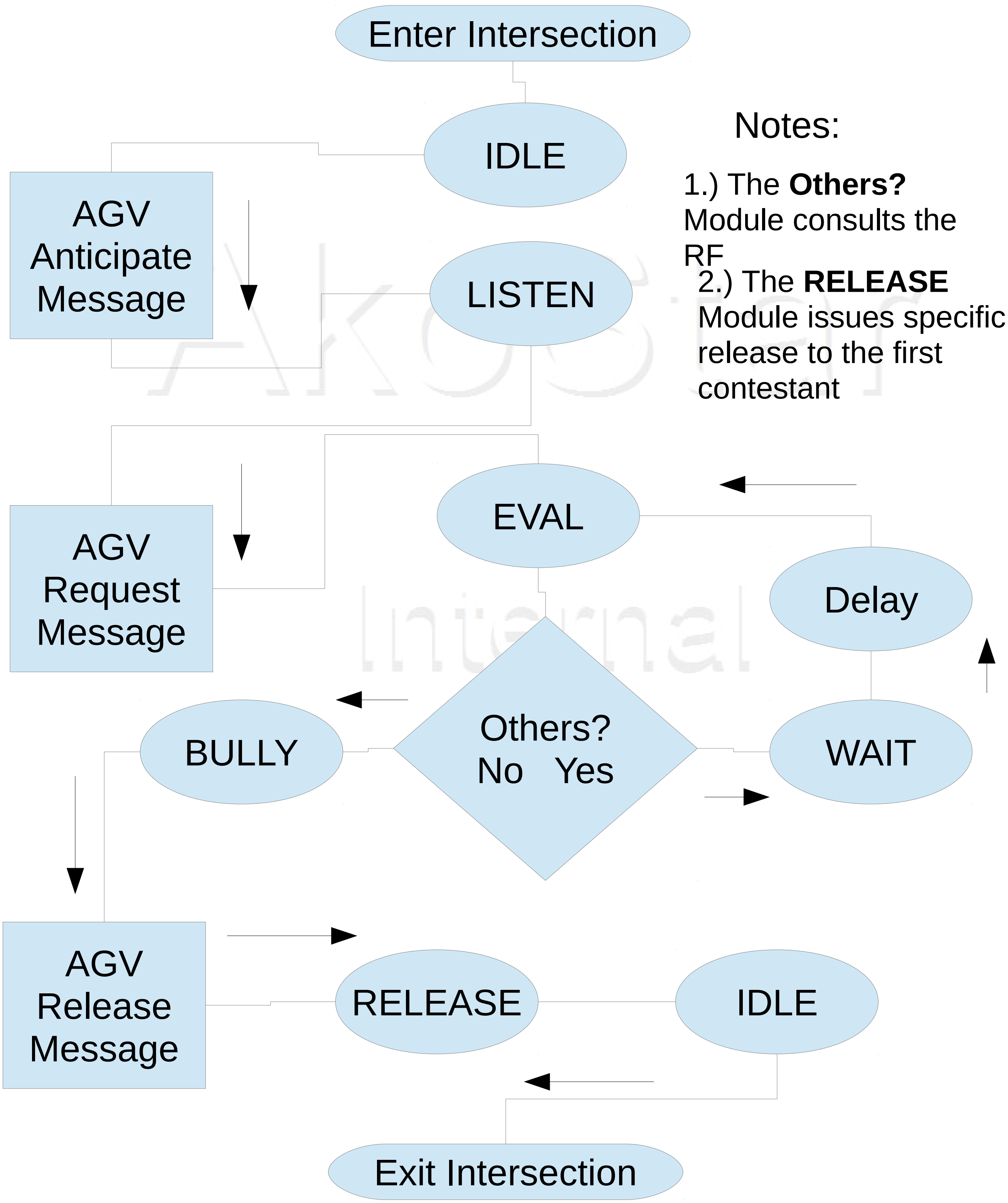
The first phase is called '*anticipate intersection*'. In this phase the QCAN2 will listen, and determine if there is a contesting AGV at the same intersection. If there is, the QCAN2 instructs the AGV to slow it's speed by transmitting on its serial port the OCCUPIED status, setting BIT_7 and BIT_6.

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 in chapter xxx. 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)

Intersection logic chart

(Implemented, informational section only. This chart represents a simplified view of events.)



Bit allocation update:

Most of the bit allocation is identical to the previous version, except Bit_2; also for completeness, I included the switch bits.

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 closed (FE context only)
Bit_0 (0x01) when set => Switch_0 closed (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 decision on re-routing.

Door controller

The the QCAN2 door controller will bridge communication between the AGV and a (fixed) door controller mechanism. The AGV communicates with the door controller via the door commands. The QCAN2 door controller has three major function groups:

a.) Control the door (open / close)
b.) Occupy Door zone
c.) Report the status of the door.

The commands correspond to functions of the door. a.) Open door b.) Close door.
The status reports of the door correspond to: a.) Door in transition. b.) Door opened. c.) Door closed.

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.

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):

This part of the page is left intentionally blank

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 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.

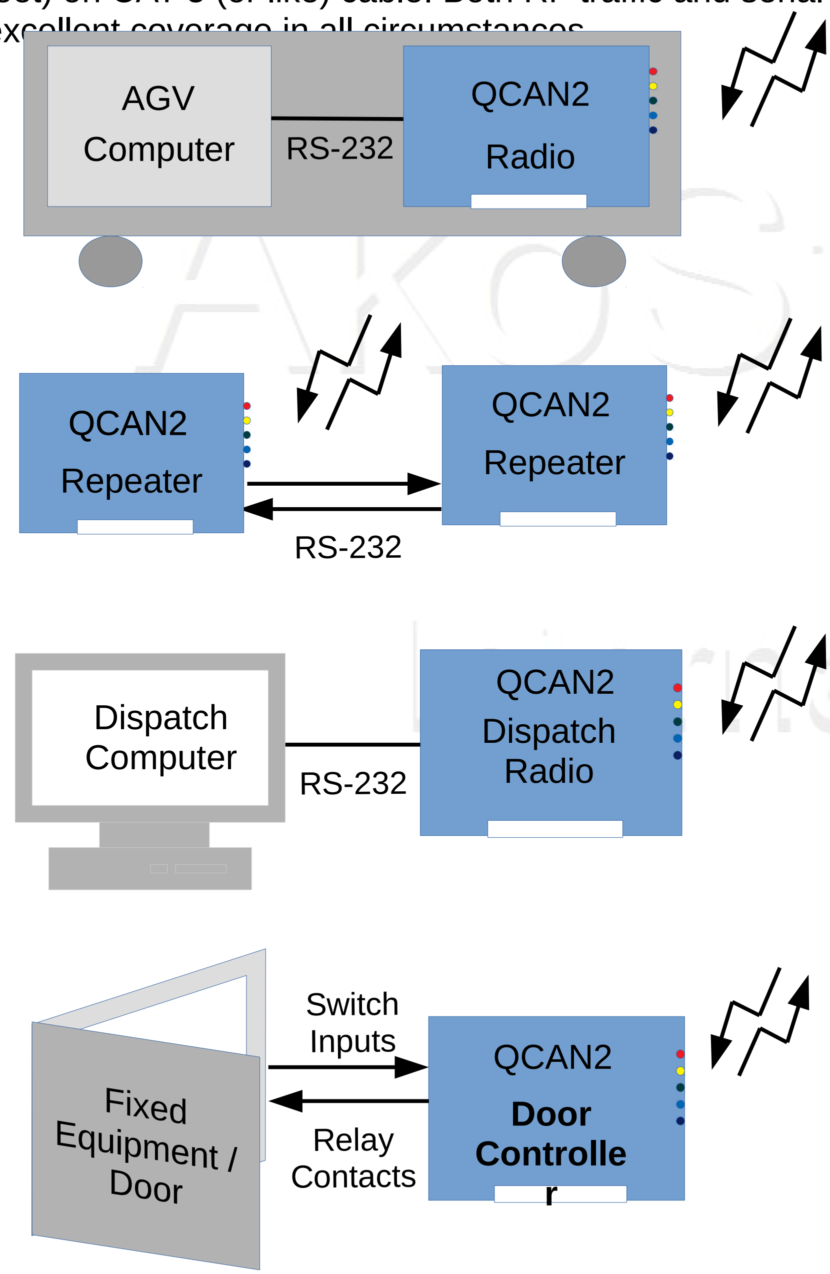
The Mac address can also then be observed as the AGV number, or zone number, or intersection number. Because the QCAN2s communicate with each other based on this Mac address, zero configuration is automatic. The other major aspect of the configuration-less operation of the QCAN2, is that every control command is unique, and pertinent to a specific function, so the QCAN2 will always be able to distinguish what action to perform. (This is addressed further in the door controller / intersection controller combo)

This part of the page is left intentionally blank

QCAN2 Dispatch & Repeater overview

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
```


This part of the page is left intentionally blank

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

DOOR (FE) Relay Controls

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. Feedback needed / welcome.

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

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.

Supplementary tools and troubleshooting

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.

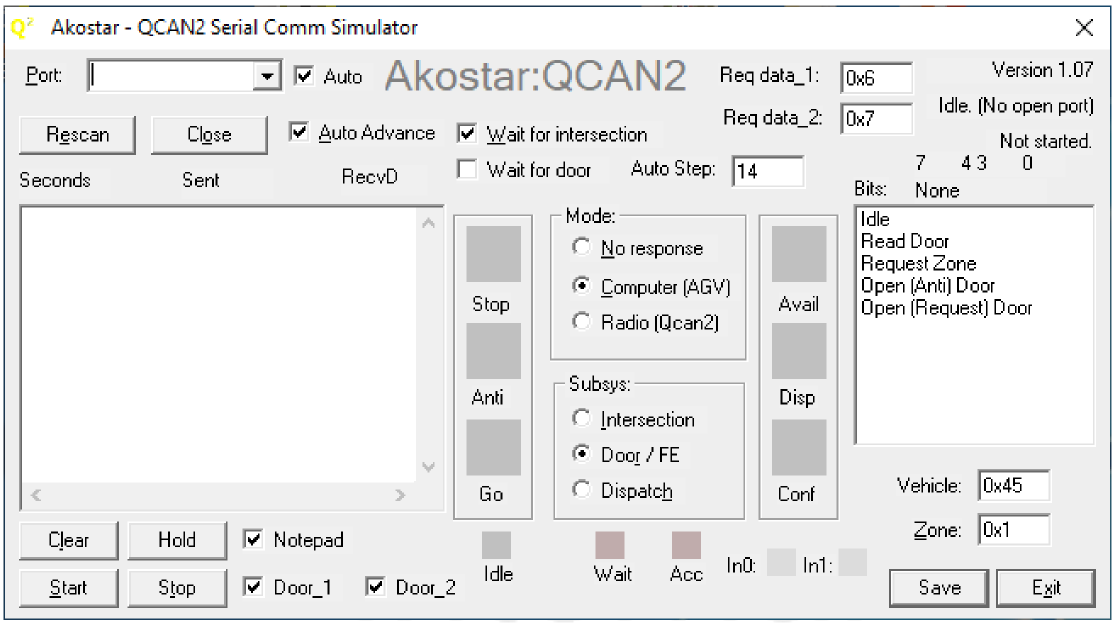
** (Putty: Open Source terminal program)

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 has some additional functionality to aid automated testing.



Updates to the Simulator:

Back-porting the QCAN2 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 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, but notepad will easily take several tens of megabytes of data. All the other features have limits testing, they can be run indefinitely, such as a regression test.

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

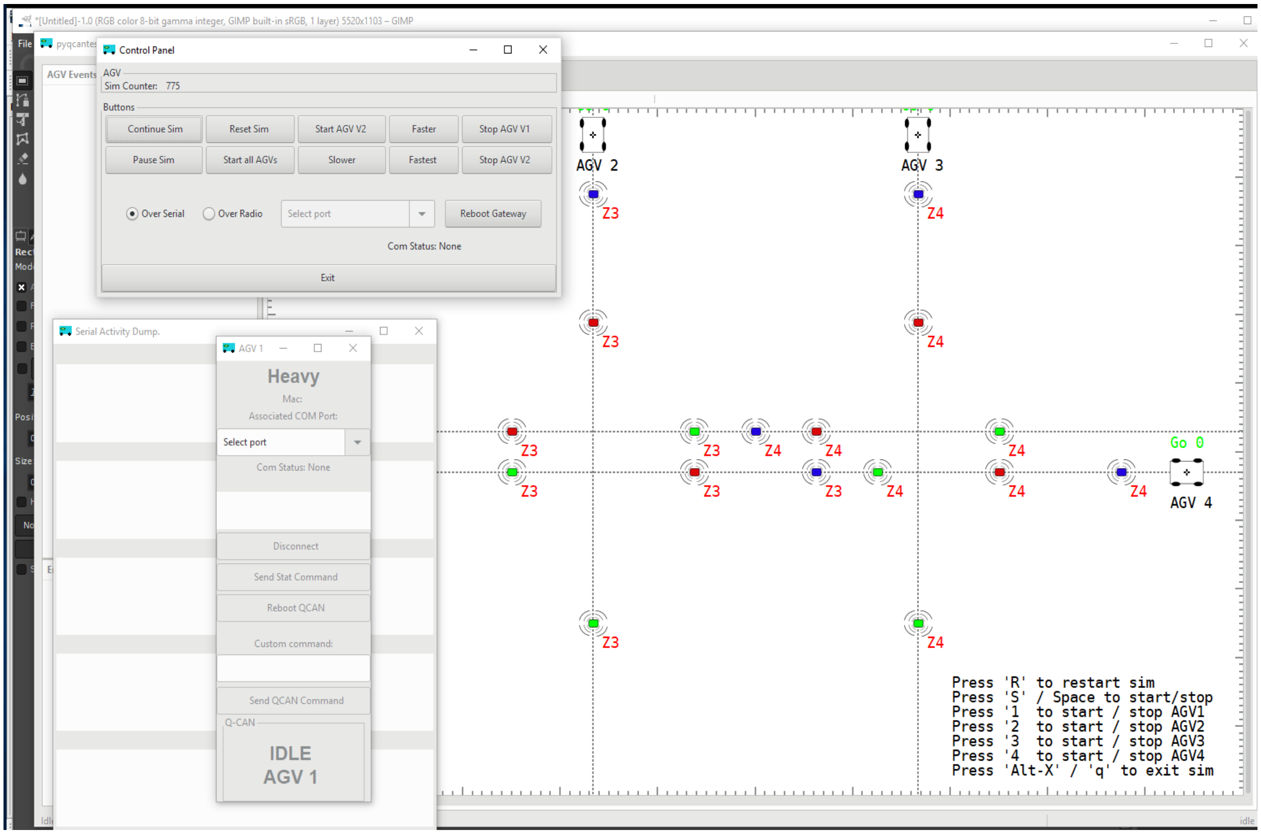
wait dispatch to -> accept dispatch negative (red)

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

Python Based Simulator

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 in the provided laptop.

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
Mar-26-2020