Welcome to the Z-Wave Summit

ABSTRACT

Z-Wave wireless Internet of Things (IoT) devices are hard to test! There are countless devices already in customers hands with bugs in them that make Z-Wave seem unreliable when in fact many of the issues are bugs in the device firmware. Eric Ryherd, Z-Wave expert and consultant, describes some of the failures that are still shipping today and best practices when testing your IoT device to reduce the chance your device fails in your customers hands. Simple command sequences sent one at a time by a test engineer is not representative of the real world packet storms that occur in an apartment building with complex RF reflections and multiple interfering RF networks. Your device has to work in the real world and to do that you need to simulate those terrible conditions that do not happen on the engineers desk.

Express Controls

Z-WAVE

OUTLINE

- Review of product failures
- 7 Rules for Testing Z-Wave devices
- Firmware guideline
- Watchdog guidelines
- Conclusion
- Q&A
  - Ask questions as we go but I may defer to later

EXAMPLES OF PRODUCT FAILURES

- Send two back-to-back Z-Wave commands quickly to door lock crashes until it sleeps and wakes
- Garage door sensor sleeps in 2s so can't get battery level if routing is required
- Sensors that spew readings clogging network
- Learn mode always on and randomly excludes
- OTA times out if any adjacent network traffic
- Doorlock bricks after firmware update
- Battery level reporting accuracy
- Low percentage nodes drop off network

Express Controls

Z-WAVE

A popular door lock would crash if you sent it 2 commands in a row – set user code followed immediately by a get user code would crash the 2nd micro on the lock. Fortunately once the lock went to sleep and woke up again it would recover.

A popular garage door sensor sends a Wakeup Notification and then in less than 2 seconds is back to sleep. If the sensor has to route even thru 1 node the delivery takes most of those 2 seconds. If the hub requests the battery level, since the Battery_Get has to be routed back to the sensor it takes again a second or so. But the sensor is already asleep. This is a classic case of the device working fine on the engineers desk where everything is in direct range but fails in the real world where delays and routing are common. The CTT now checks that a device is awake for approximately 10s.

A popular energy sensor in it's first releases would blast out Z-Wave commands every fraction of a second if the energy readings were varying. Resulting in an unusable z-wave network and an unusable product. The product works fine if it is measuring the energy drain of an incandescent light bulb, but put it in the real world and not only does the product itself fail, it makes the entire Z-Wave network fail.

My own products are of course not immune from failure. This is my EZMultipli multi-sensor and for the first seven revisions of the firmware and shipped several thousand units, the sensor would remain in the Z-Wave Learn Mode even after the learn mode expired. Tricky to find as the device functions completely fine. But put it in the real world and a user goes to exclude another device and for some reason the EZMultipli would magically be excluded and is no longer part of the network. Fortunately it has a

color led that blinks aqua when it is not part of a network so the user immediately knows something is up.

That wasn't the only failure of this device, it also had a timeout for the OTA firmware download that again worked fine on the engineers desk, but put it in the real world with an adjacent Z-Wave network causing delays and collisions would cause the timeout to trigger making firmware updates almost impossible. The "solution" on several blog sites was to take the device to your grandparents house and update the firmware there and it would work just fine.

IoT failures are not limited to Z-Wave. AirBnBs partner LockState found that occasionally a lock would brick after a firmware update of their Wifi door lock. When failures occur randomly and at a relatively low rate (less than 10% of the time) are particularly hard to find.

Battery level reporting in general is a common source of error. Some devices will still be working fine at 10% battery power or even 0. Others get very flakey at even 50%. Understand that measuring battery levels is really difficult given the many different types of batteries – lithium, alkaline, rechargeable. The big problem is that you cannot emulate weak batteries with a power supply set at 2.8V instead of 3.2V. Weak batteries voltage will drop under load more than full batteries. Use real weak batteries and not a power supply.

Z-Wave is still plagued by reports for nodes randomly dropping off the network. The number is small enough that it so far has eluded my ability to capture or even observe the failure but I get reports from several high volume customers that this is an ongoing problem.

7 RULES FOR TESTING Z-WAVE DEVICES

1. Written Test Plan/Script
2. Sigma CTT – checks the FORMAT of Z-Wave
3. Test across temperature and voltage
4. Test in the real world
5. Build complex Z-Wave networks
6. Test with other Hubs and devices
7. Automate testing for repeatability

Express Controls

WRITTEN TEST PLAN

- Functional vs. Production plans
  - Production may reuse a small subset of Functional tests
- Setup & Equipment
- What will and WON'T be tested
- Procedure
  - Action
  - Observe expected operation
  - Pass/Fail requirement
  - Linkage to Specification Requirement
- Operator or computer?

Express Controls

Z-WAVE

This talk is discussing the Functional Testing of a product – making sure the product works. A subset of the functional test plan is often used for the Production Test Plan which is merely looking for manufacturing defects to screen out and repair defective units.

A typical Written Test Plan consists of:

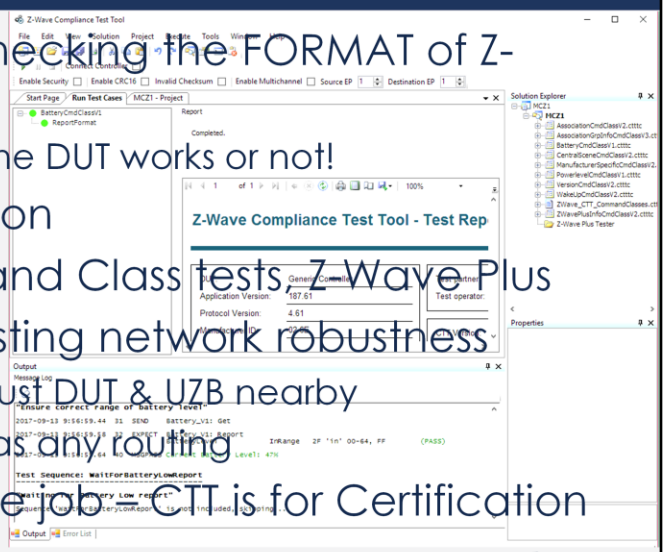
- The required equipment list and how to set it all up
- A high level description of the test plan and specifically what will NOT be tested
  - Most common limitation is the range of configuration settings – testing every possible setting might be impossible.
- A list of steps to test various parts of the design
- Each step should list:
  - The action needed to be performed – for example, press the LEARN button
  - Observe the expected result of the action – The LED should blink and a NIF is sent out and inclusion beging
  - Pass/Fail requirements – Device PASSes if LED blinks and the node joins the network – failure codes for various types of failure (LED on solid instead of blinking, fails to join, fails security etc.
  - Ideally every test has some connection back to a specification for the product itself. There are many tools for helping with this. At a minimum a comment on the PURPOSE of the test it
- A very important part of the test plan is will it be run by a human operator or

largely automated using a computer such as the RPi

- Human operator is easier and quicker to develop
- Automated testing takes longer to develop but is then easily run on each release and can be run on multiple DUTs looking for marginal issues
- Either way the test plan has to track the firmware development and changes in the firmware requires changes to the test plan and scripts

CERTIFICATION TEST TOOL CTT

1. CTT is really good to checking the FORMAT of Z-Wave commands
   1. But does NOT check if the DUT works or not!
2. Required for Certification
3. Has two parts: Command Class tests, Z-Wave Plus
4. CTT is NOT useful for testing network robustness
   1. Typically runs best with just DUT & UZB nearby
   2. Usually fails if network has any routing
5. Use the right tool for the job. CTT is for Certification



Express Controls

Z-WAVE

If you pass the CTT then you'll pass Z-Wave certification so the product is ready to ship!
NOT!
The CTT is an invaluable part of the test plan. But it is only one small piece.
The CTT is quite good at finding errors in the FORMAT of the Z-Wave commands.
But it has ZERO knowledge of your application and if your product functions as desired or not.

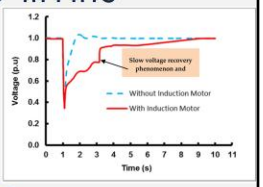
Make sure you run both parts of the CTT! Command Class tests and the Z-Wave Plus tests.

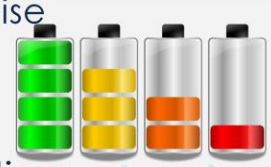
CTT scripting language is brittle and will fail if the wrong frame is received at the wrong time. It works best in an isolated RF environment.

CTT is for certification. You can use it to develop your own custom scripts for functional testing but because of the limited abilities of the scripting language there are better solutions available.

VARY ENVIRONMENTAL CONDITIONS

- Vary the product power supply at extreme limits
  - Strange things happen at the margins
  - Exposes weak or marginal conditions
  - Test with multiple DUTs
  - If wall powered, test extended brownout & noise
  - If battery powered, use real weak batteries!
- Vary temperature
  - Thermal chambers are expensive and limited time
  - Freeze spray/heat gun works good enough





Express Controls

Z-WAVE

Some EEPROMs only function down to 2.7V but the Z-Wave chip will operate down to 2.4V but then since the EEPROM doesn't work Z-Wave fails. Running the device at the margins of voltage and temperature will identify these issues before shipping the product.

Not everyone has a thermal chamber at their disposal. But a can of freeze spray and a heat gun are easy to use. Obviously a thermal chamber is recommended.

REAL WORLD TESTING

- The engineers desk is not a valid testing environment – no routing – no dropped frames
- Use the target hub in the target application with extra checks on every communication
  - Keep a Zniffer or CIT running
  - Check validity of every frame
    - Is sensor accurate or at least in range
    - Has every scheduled reading taken place
    - Put 2 or more units in the same place and cross check
- Vary building materials and RF noise environment

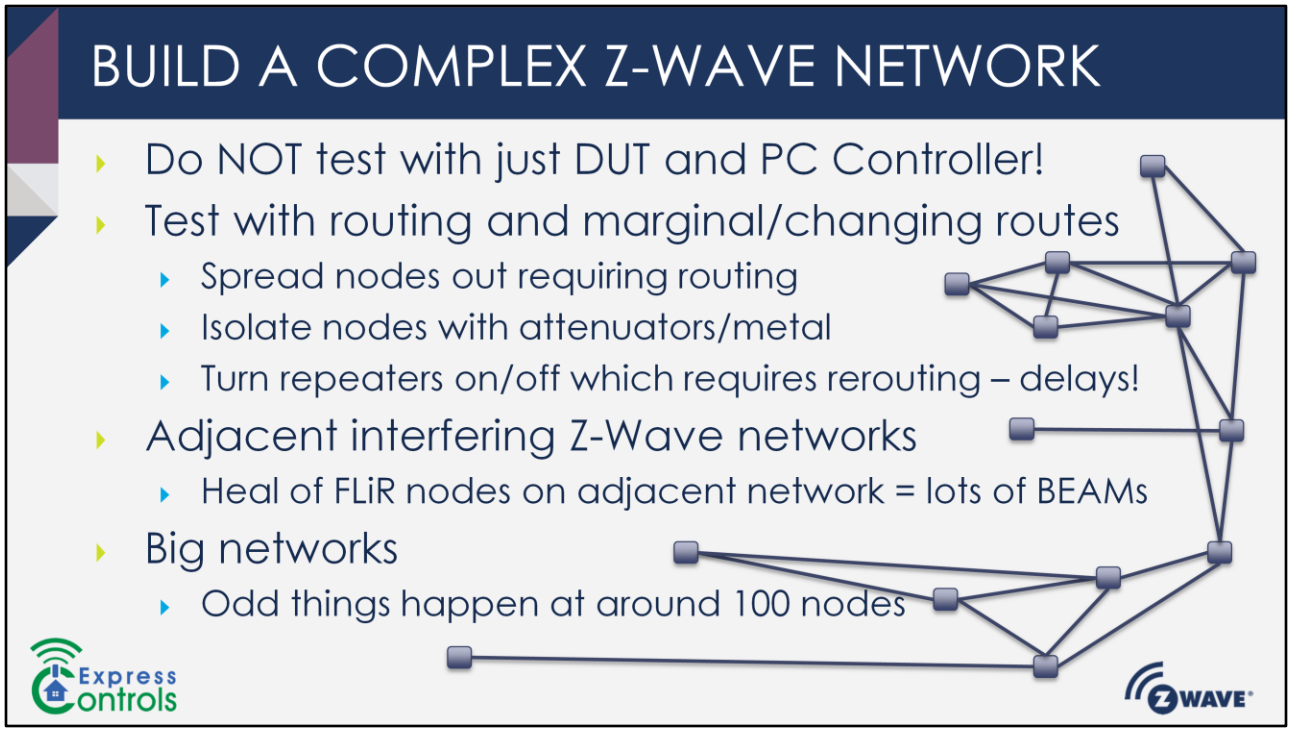
Express Controls

Z-WAVE

The most common problem I keep seeing in so many products is they work fine on the engineers desk with just a couple of nodes and no routing.
But put the device out into the wild, with strange RF reflections, varying RF conditions as humans walk around or doors open/close, or new nodes are added to the network or die or are removed, and the device fails or worse yet makes the entire Z-Wave network fail.
Does your device work in multiple real-world environments?

Every engineer on the project should have a couple in their home or apartment and connect them to something fairly obvious so if it's not working properly you know immediately.
If you have to power cycle – THERE IS A BUG!
If at all possible have a Zniffer running all the time (or at least most of the time).
You'll find all sorts of interesting things! Like valid frames that are missing a byte!
Then the temperature reading is as hot as the surface of the sun or cold as pluto! This happens with the 9.6K and 40K frames since they only have a simple 1 byte checksum.

The varied building materials and marginal conditions are virtually impossible to duplicate in a lab environment. So get out there and install some units!



Build complex networks which require routing and have marginal, changing RF conditions.

It is difficult to simply spread nodes out unless you have a really large office building. Z-Wave often easily reaches more than 40m which in many cases is larger than your office building.
It is possible to create marginal RF conditions using metal boxes that limit the range of nodes.
Deliberately detune the antenna so the range is only 10' or so which then means the node has to route to a neighbor to get to any moderately distant node.

Create varying routing conditions by plugging repeaters into Z-Wave switches then turning them on or off which requires the routing to change.
One of my favorites is to place a Z-Wave power strip with repeaters in each outlet. Place the strip mid-way between two sets of nodes. Then turn the individual outlets on/off to force routing changes.
Interfering adjacent networks is easy to create and common in virtually all of our offices. Healing a network with a bunch of FLiR nodes results in tons of WakeUp Beams which means lots of RF interference.

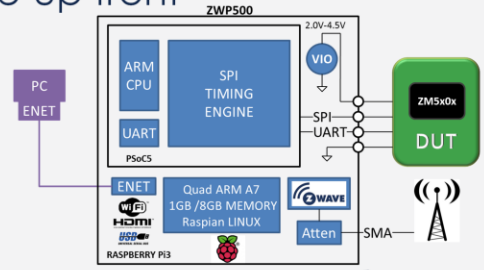
Big networks result in their own issues as joining starts to take a really long time. Will the hub support timeouts long enough to actually build a 100+ node network?

It can take hours to build a 100+ node Z-Wave network. As the number of nodes increases every node has to check if every other node is a neighbor which takes longer and longer.

Every Hub vendor will send a different order of commands during inclusion/exclusion and daily operation. Some constantly poll, others send incorrect commands. They all have bugs.
Using these hubs will make sure yours is robust and meets the real-world requirements.
You can also work with the hub vendor to properly support your device.
You don't have to become an expert in their interface. Most are simple enough for soccer moms so you can figure it out too!

AUTOMATE TEST PROCEDURE

- Human Operator vs. Computer?
- Computer can press buttons with precision
- Computer can repeat tests all weekend long
- Camera can check menus, LEDs and screens
- Automated testing takes more time up front
- ZWP500 Raspberry Pi Based programmer/tester platform



Express Controls

Z-WAVE

The test procedure originally always starts out being performed by a human operator. The goal is to automate as much of the procedure as time allows (which there is never enough time).

There are a lot of things a computer can do that a human can't

A computer can press buttons with millisecond precision and measure the result of each to verify timeouts between a TAP and a PRESS-AND-HOLD

A computer doesn't sleep – it can test all night, all weekend, all week without a break

Each release of firmware can be re-tested typically in just hours by the engineer responsible for the release

Automated testing takes more time to develop up front. And of course if the functionality changes, the automated testing program also has to change so it is best not to start until the development is partially frozen.

Express Controls has an automated Z-Wave Testing platform called the ZWP500. A Raspberry Pi3 based Linux computer is augmented with a dedicated real-time microcontroller to program and test a Z-Wave based IoT device.

FIRMWARE CODING RECOMMENDATIONS

- Code with failure in mind
  - What happens if the hardware bit NEVER gets set?
- Avoid WHILE statements
  - Use FOR with a long timeout
- `IF timeout>=10` instead of `IF timeout==10`
- Reboot on >10 failures or zero comms in 24hrs
- KISS – Keep It Simple!

Express Controls

Z-WAVE

IoT devices must run for YEARS without crashing or locking up.

Code with failure in mind. What will happen if a FIFO never goes empty or the UART never becomes free? Your code needs to keep on running or – it needs to reboot and hopefully the issue will clear itself up.
Nothing is ever perfect. There are high-speed particles traveling thru us right now damaging the silicon in your phone right now. The same thing will happen to the Z-Wave chip in your product eventually.
Code with failure in mind to avoid getting stuck. Reboot to potentially clear up the failure so the customer never knows. We'll talk about watchdogs in the next slide.

WHILE statements are just one step away from FOREVER loops. Far better to use a FOR loop so eventually, the loop will be exited even in the event of a minor hardware failure.

When testing variables, compare with a greater than instead of just a simple equality is safer. The classic example is a timeout – suppose a timer is counting up once per second. Comparing with ==10 means that if somehow there is a failure and timeout somehow skipped over 10 and went to 11, then the timeout won't happen for another 4 minutes if timeout is a byte or 7 DAYS if it is a word or virtually forever if it is 32-bits!

A new tactic I am now apply to all of my projects is to reboot (via watchdog timeout) if there have been more than 3 frame delivery failures in a row or in general no comm traffic in over 24 hours. Something is amiss if we're not talking so why not reboot and hope things clear up.

Keep the code simple! The more complex the code, the more bugs. PERIOD.

WATCHDOG BEST PRACTICES

- ALL PRODUCTS SHOULD ENABLE THE WATCHDOG!
- Disable WatchDog during development
  - Want device to get "stuck" so it can be fixed
  - Do NOT ignore occasional power cycle
- Calling `ZW_WatchdogKick()` in ApplicationPoll is NOT enough!
- Test Watchdog code
  - #define a small block of code to force failure
  - Trigger with a special BASIC_SET of 0xDE
- Log the number of Watchdogs
  - `If (ZW_WAKEUP_WATCHDOG==WakeUpReason) WDogCnt++;`

Express Controls

Z-WAVE

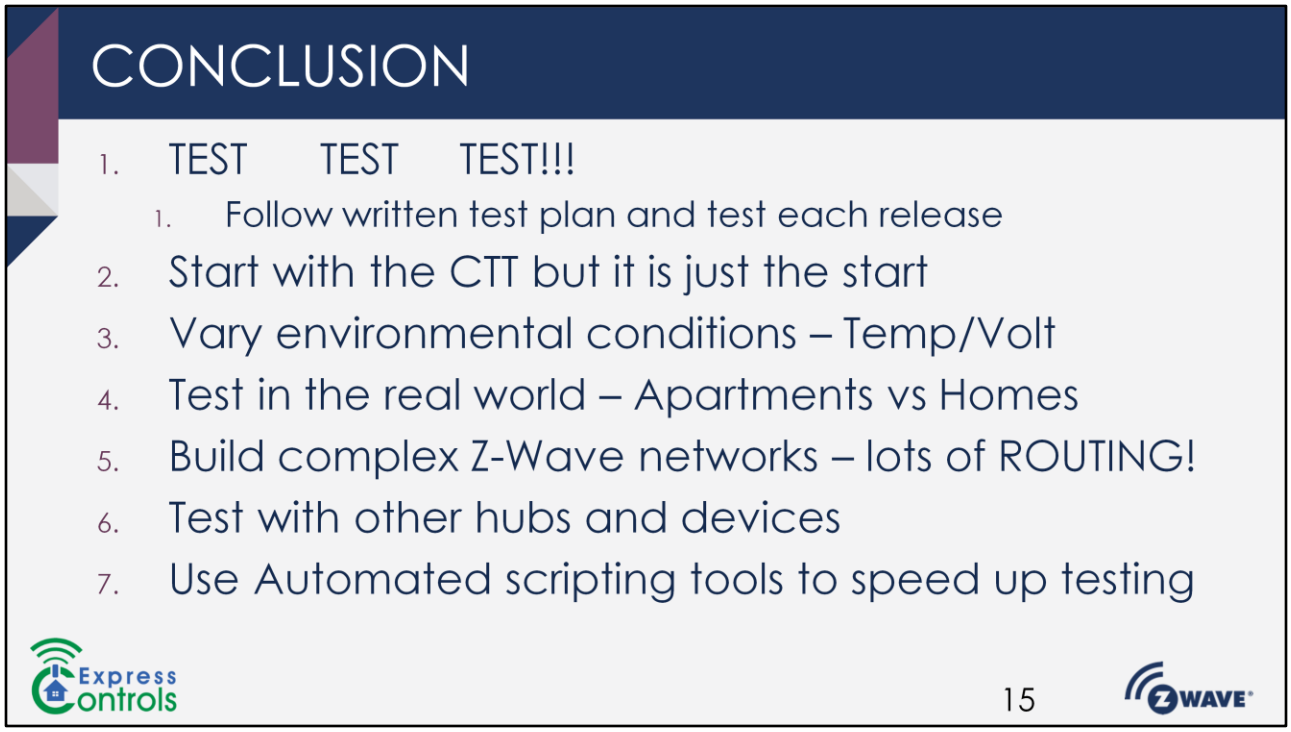
The WatchDog is a critical part of keeping an IoT system running in the presence of failures.
We never want a device to get STUCK and become unresponsive. With the Watchdog enabled, it is much less likely.

During development however, we WANT the device to be STUCK. The stuck state is really obvious that something has failed and is usually easier to debug. Rebooting often masks failures.
Realize that the device being STUCK is a FAILURE. Having to power cycle is BAD and must be investigated.

The algorithm to "kick" or the more politically correct "pet" the WatchDog is NOT something to take lightly but needs a significant amount of thought and some experimentation.
The sample Z-Wave applications simply call KICK every ApplicationPoll. While this is at least a minimal solution, a lot of other failures might prevent your device from working but ApplicationPoll is humming right along.

Testing the Watchdog is a challenge. My solution is to put a small amount of code in #defines that an invalid BASIC_SET (0xDE for example) triggers the watchdog. The BASIC_SET will force a condition to be set that should trigger the watchdog. Make sure it does! Untested code is code that does not work!

I have been logging the number of Watchdog resets in NVM which is handy during development to check and see if the watchdog is triggering when not expected.
You need a special configuration register or something to read it out. Or tack on some extra data to some other command.



1. Have a plan and write it down
    1. Add to the plan when interesting failures are found
    2. Review plans from previous projects on new projects
2. The CTT is part of any Z-Wave product test plan – but it ONLY tests the FORMAT of the Z-Wave communication
    1. The CTT does NOT test that the data is valid or accurate
    2. The CTT is low bar – it is not the end of testing but just the very beginning
3. Voltage is easy to vary
    1. Funny things show up when the voltage is marginal
    2. Temperature is a little tougher as it requires an expensive thermal chamber, but a can of freeze spray and a heat gun are easy and cheap
    3. What are the sensor and battery readings at the extremes?
    4. Are switches/relays operational at the extremes
    5. If wall powered, what happens during an extended brownout and the DUT is under maximum load?
4. Test in the real world – put a couple of these devices in your home – or every engineers home!
    1. Apartment vs. single family – different environments and building materials.
    2. Adjacent Networks – interference – marginal communication

    3. Use a real hub which has buggy software!
1. Build Complex Z-Wave Networks
    1. Spread out some nodes and make them route!
    2. Can you still OTA? Get battery levels? Change configuration parameters? Get firmware revision?
    3. Adjacent networks result in highly variable conditions and occasionally failure to deliver – does the APPLICATION still work?
    4. What happens when you can't get the message thru because an adjacent network is doing a network heal on a bunch of FLiR devices?
2. Test using other Hubs and devices
    1. Every hub has bugs. Does your device still work at least at some level?
    2. Every hub will do some strange things – setup the lifeline and association groups in different order – send the same command many times – rapid fire series of commands
3. Custom Scripting and Tools
    1. Ideally the test plan is mostly automated with minimal amounts of manual user interaction
    2. Ideally you can rerun the full test suite on every incremental release because it is fully automated
    3. Express Controls has a product – the ZWP500 – and the expertise to help out in this area.

QUESTIONS?

Eric Ryherd (aka DrZWave)
Express Controls LLC
+1 (603) 889-4841
Eric@ExpressControls.com
www.ExpressControls.com
DrZWave.blog

Z-Wave Consulting, Training and Product Development

Express Controls

16

Z-WAVE

Fireside chat with Eric - questions and answers and a discussion with the entire group

AUTHOR BIOGRAPHY

Eric Ryherd licensed Z-Wave in 2003 to develop IoT devices before the term IoT even existed. Light switches, motion & temperature sensors, water valves and meters, hubs, window shades, remote controls are just a sample of the Z-Wave IoT devices developed and tested by Express Controls. Eric applies his Z-Wave expertise in consulting, training and assisting with Z-Wave Certification to companies of all sizes. Read more by Eric at his blog – DrZWave.blog.