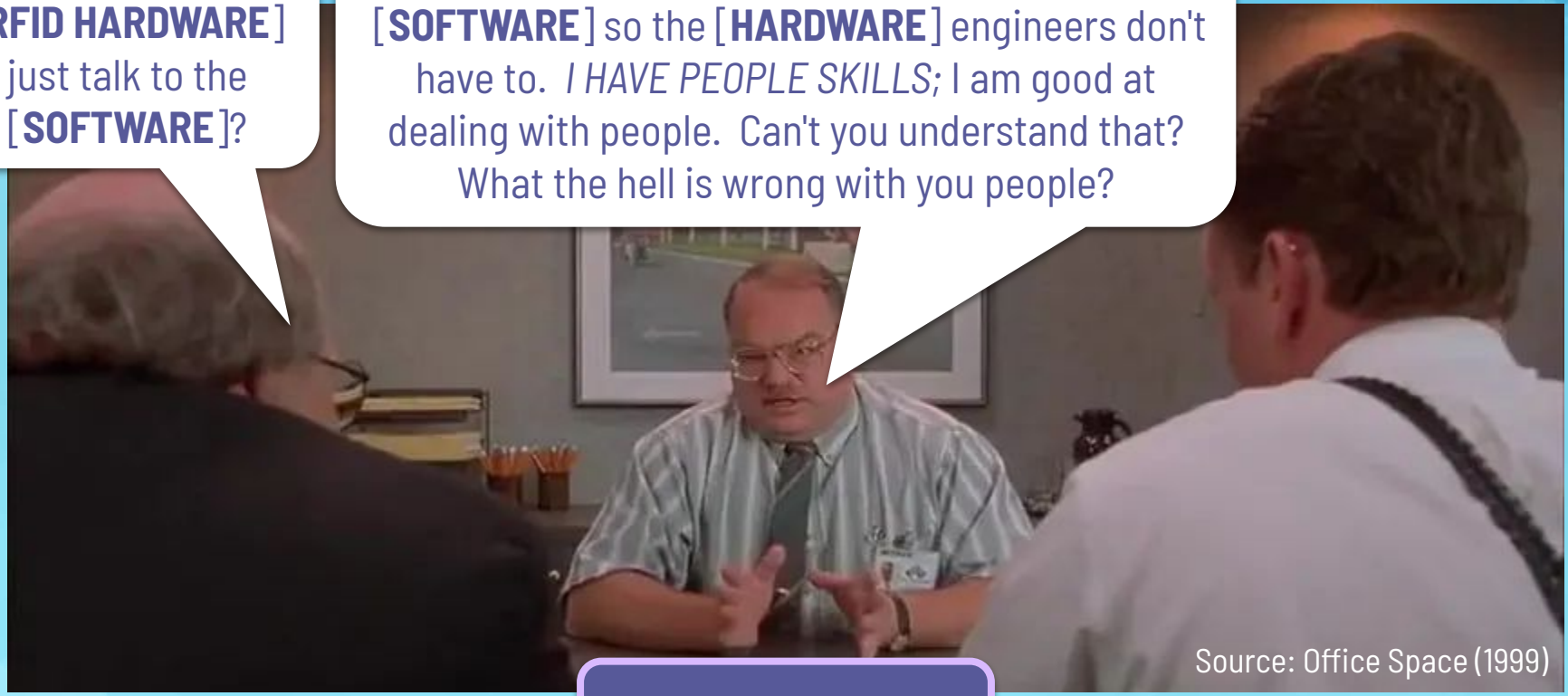
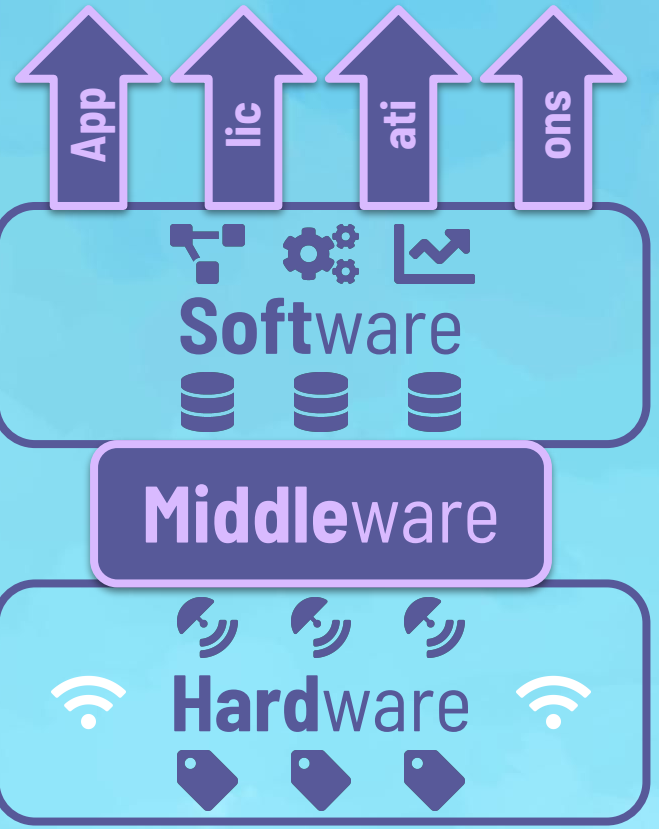
Middleware & Applications

IEEE RFID 2026 Tutorial presented by **Jeffrey Dungen**


Why can't the [RFID HARDWARE] just talk to the [SOFTWARE]?
Well—well look. I already told you: I deal with the [SOFTWARE] so the [HARDWARE] engineers don't have to. *I HAVE PEOPLE SKILLS;* I am good at dealing with people. Can't you understand that? What the hell is wrong with you people?
Source: Office Space (1999)

Middleware

Hard-Middle-Soft, then Applied...



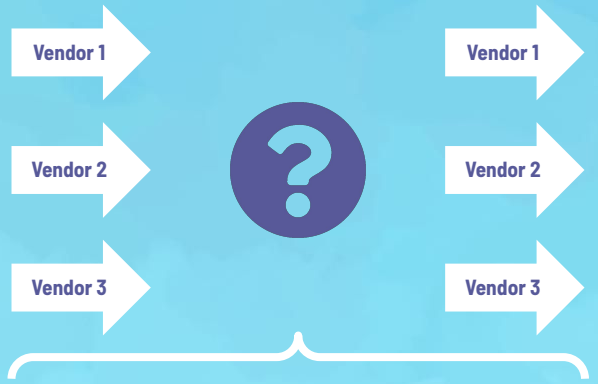
Remember everything that **Dobkin**, **Durgin** and **Degnan** talked about?

Why do we need Middleware?

Middleware is like a universal ***translator*** between hardware & software.

Standards observed over-the-air:

- ➔ RAIN RFID
- ➔ Bluetooth Low Energy
- ➔ etc.



Vendor-specific interfaces...

Frustration for integrators and users developing applications *without* middleware.

Appreciate the app developer

The people developing the **application** layer:

- ***Don't* care** about how the hardware works.*
- ***Don't* care** about how the protocols work.*
- ***Do* care** about having **usable data** in a format they understand.

* sorry Dan, Greg and Brian!

Let's tidy up the messy middle and make them happy!

Part 1:

The Hardware Side of Things

Hardware and Protocols

Protocols: RAIN RFID

There's a standard for the wireless interface:

- ➔ **EPC UHF Gen2 Air Interface Protocol** (ISO/IEC 18000-63)*
- ➔ Enjoy reading: **https://ref.gs1.org/standards/gen2/**

* presented in Dan's tutorial

There's a standard for identifiers and data:

- ➔ **EPC Tag Data Standard** (TDS)*
- ➔ Enjoy reading: **https://ref.gs1.org/standards/tds/**

Tomorrow:
The ID in RFID Workshop by GS1

* we'll come back to this later

Protocols: Bluetooth Low Energy

There's a standard for the wireless interface and more:

➔ **Bluetooth Core Specification** (currently v6.2)*

➔ Enjoy reading: **https://www.bluetooth.com/specifications/specs/core-specification/**

* it's only 3897 pages long!

There are complementary specifications for identifiers and data:

➔ Read them all: **https://www.bluetooth.com/specifications/specs/**

Protocols: Bluetooth Low Energy

Rather read an IEEE RFID tutorial than a standard specification!?!

https://2017.ieeerfid.org/files/2017/01/IEEE-RFID-2017-BLE-as-Active-RFID.pdf

Protocols: but wait, there's more...

Other wireless protocols used around the world include:

- ➔ **UWB** (Ultra WideBand): lots of different flavours 🙄
- ➔ **NFC** (Near-Field Communication): short range, broad adoption in mobile
- ➔ **EnOcean** (ISO/IEC 14543-3-10): wireless sensor networks
- ➔ and many others...

Do they each have their own unique specification? *Of course!*

Readers decode RAIN RFID

I *could* observe standards, but chances are, I'll relay data in a **vendor-specific** format.

LLRP and RCI: RAIN has standards!

Low-Level Reader Protocol (LLRP)

- ➔ Widely supported by readers.
- ➔ Legacy protocol (2007).*
- ➔ Enjoy reading:
  https://www.gs1.org/standards/epc-rfid/llrp/1-1-0

* remember XML!?!

RAIN Communication Interface (RCI)

- ➔ Low adoption.
- ➔ Modern protocol (~2018).
- ➔ Latest version may not even be publicly available? 🤷

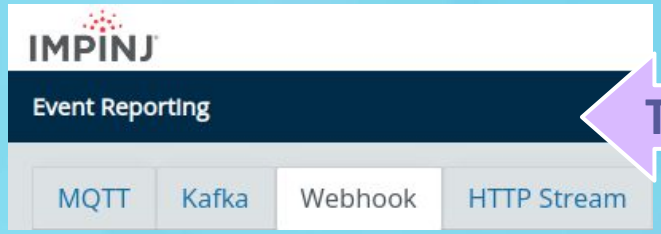
Our generation speaks **LLRP**!

I (alone?) speak **RCI**!

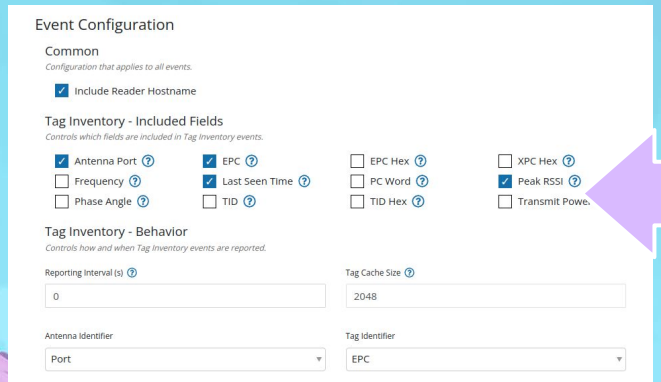
Clairvoyant Technology S2 Reader

Reader configuration examples

Impinj R700 web interface:



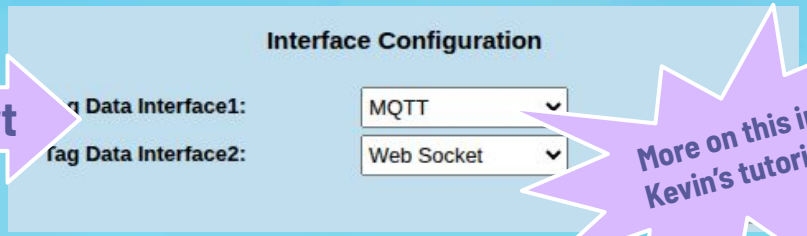
Transport



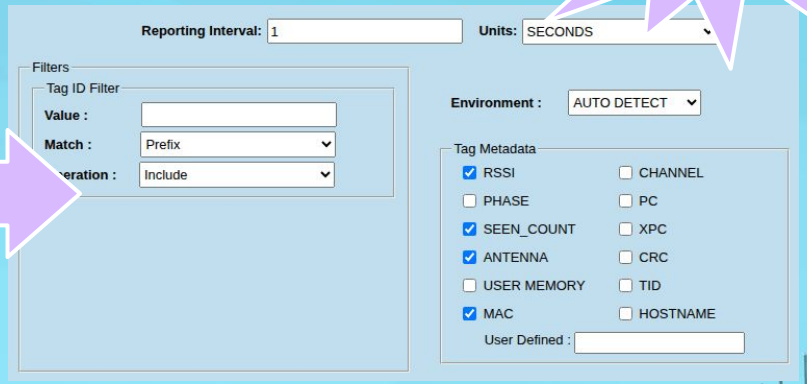
Data

https://reelyactive.github.io/diy/impinj-r700-config/

Zebra FX9600 web interface:



More on this in Kevin's tutorial

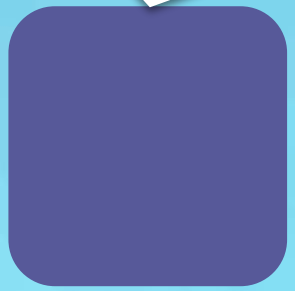


https://reelyactive.github.io/diy/zebra-fx9600-config/

Gateways decode BLE

No standards? No worries!
I'll relay data in a
vendor-specific format.

BLE Beacon



BLE **Gateway**

Vendor-specific

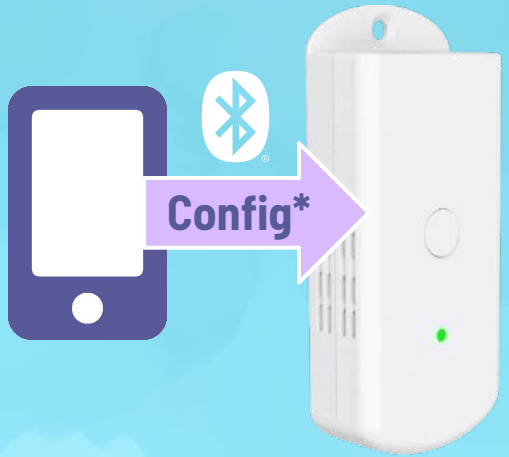
Vendor-specific

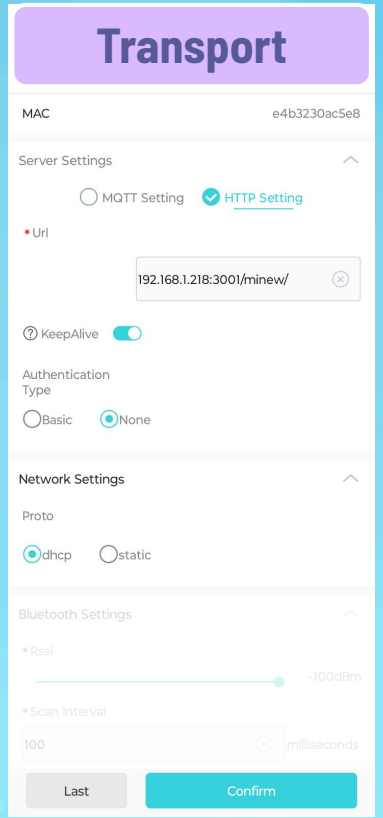
Vendor-specific

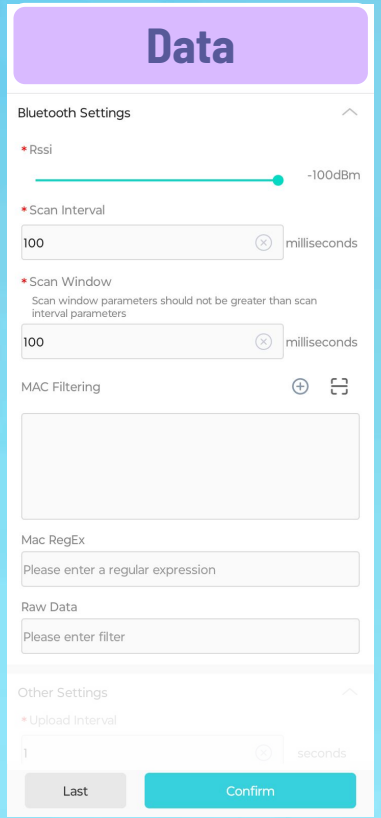
The Bluetooth SIG never created a standard equivalent to LLRP or RCI.

Gateway configuration example

Minew MG7 mobile app interface:







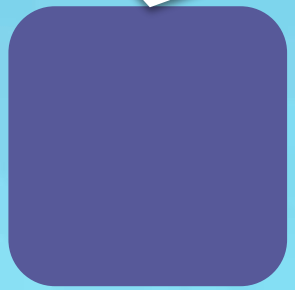
* Fun fact:
many BLE Gateways can be configured using BLE!

https://reelyactive.github.io/diy/minew-mg7-config/

WiFi Access Points decode BLE

Many of us include a BLE radio which allows us to act as ***ubiquitous*** BLE gateways.*

BLE Beacon



WiFi **AP**

Vendor-specific

Vendor-specific

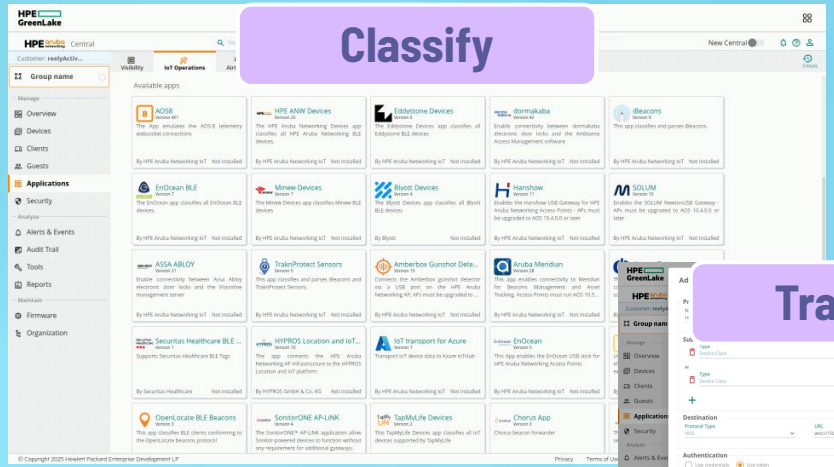
Vendor-specific

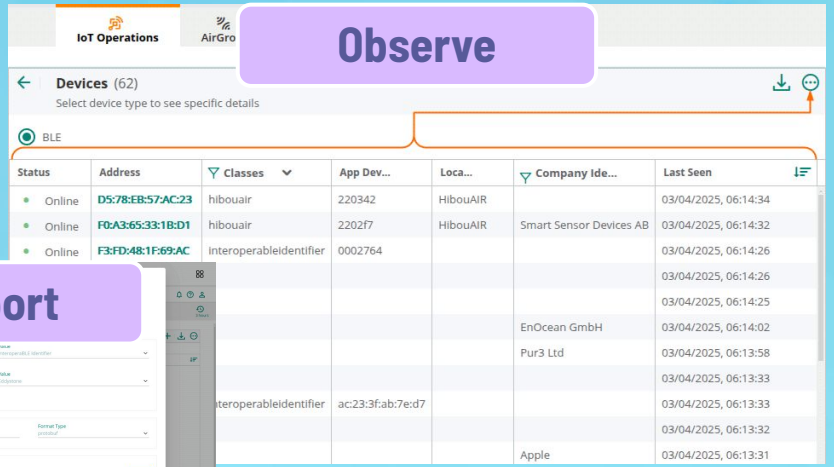
Again, there is no single standard to structure and transport the data.

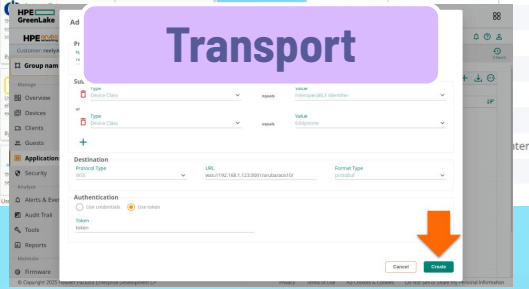
* 2024 Tutorial: Towards Ubiquitous RFID Infrastructure
https://www.reelyactive.com/science/ieeerfid/reelyActive-TowardsUbiquitousRFIDInfrastructure-240604.pdf

AP infrastructure configuration

Example: **HPE Aruba Networking Central** cloud platform







https://reelyactive.github.io/diy/aruba-iotops-config/

So how do we receive data now?

JSON via Webhook

RAIN Communication Interface (RCI)

Low-Level Reader Protocol (LLRP)

STOMP over WebSocket

JSON via MQTT

Binary via Webhook

Binary over TCP

Protobuf via MQTT

Binary serial stream

Binary over UDP

Protobuf over WebSocket

How'd we prefer to receive data?

Single, common interface for software & applications.

That's why middleware matters

Standard over-the-air. **Non-standard** post reader/gateway/AP.

Middleware bridges the gap between
what **hardware** *provides* and
what **software** applications *expect*.

Part 2:
Dealing with Data

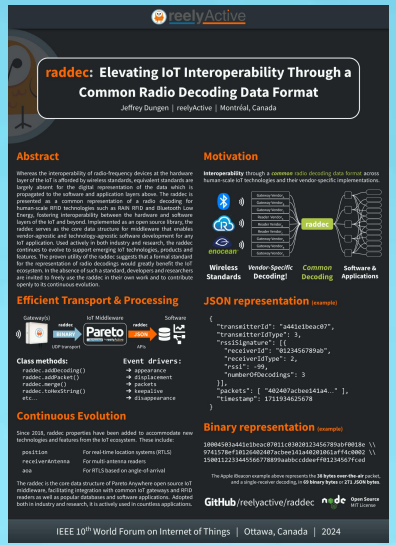
A common read? Indeed!

Is it nonetheless possible to define a **standard** way for readers, gateways and APs to represent RFID decoding data??? **YES!** 🤓

IEEE RFID 2019:

https://www.reelyactive.com/science/ieeerfid/reelyActive-CoLocatedRFIDSystemsUnite-190402.pdf

IEEE WF-IoT 2024:



https://www.reelyactive.com/science/reelyActive-IoT2024.pdf

Radio decoding: common stuff

I have a unique **identifier**.

I too have a unique **identifier**.

I'm received at a given **signal strength**, at a given **time**.

Common properties: “raddec”

A **rad**io **dec**oding (**raddec**) includes:

- ➔ Transmitter identifier
- ➔ Receiver identifier
- ➔ RSSI (received signal strength indication)
- ➔ Timestamp
- ➔ An *optional* payload and other metadata

regardless of the underlying technologies, protocols and vendor(s).

Open-source implementation:
https://github.com/reelyactive/raddec

What's in payloads and memory

Sometimes there's more to RFID than just ID.

RAIN RFID tags may have **memory**:

- ➔ memory can be *read*
- ➔ can be interpreted as sensor data
- ➔ memory may be written too

BLE devices may expose **services and characteristics**:

- ➔ these too can be *read*
- ➔ can include sensor data
- ➔ may be written too

BLE devices can also spontaneously transmit packets with user-defined payloads.

A tale of temperature (RAIN)

There's a standard for temperature data, specified in **Section 8** of the **EPC® Radio-Frequency Identity Gen-2 UHF RFID Standard**:

GS1

EPC® Radio-Frequency Identity Generation-2 UHF RFID Standard

Table 8-1: Snapshot Sensor data formats

Sensor type	Sensor	Units	Sensor data
0001_2	Temperature	°C	12-bit signed integer

Table 8-2: Snapshot Sensor data values

Sensor type	Sensor	Scale factor	Minimum value		Maximum value		Error value
			binary	scaled	binary	scaled	
0001_2	Temperature	0,0625	100000000001_2	−127,9375	011111111111_2	127,9375	100000000000_2

21°C = **336** x 0.0625

000101010000

https://ref.gs1.org/standards/gen2/

A tale of temperature (RCI)

There's a standard representation of temperature data, specified in **Annex H** of the **RAIN Reader Communication Interface Guideline (v5)**:

Sensor	Sensor name	Units	Sensor value
Temperature	TempC	°C	JSON number

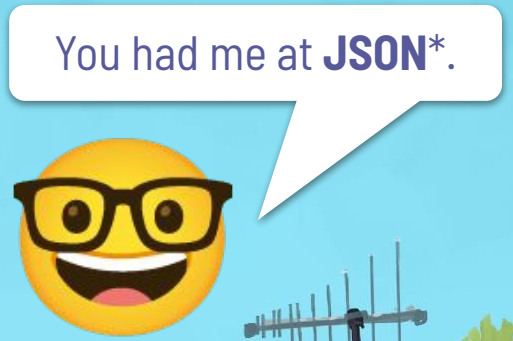
Enable snapshot sensor data interpretation in the spot profile:

```
{
  "Cmd": "SetProf",
  "InterpretData": [ { "SNAPSHOTSENSOR": null } ]
}
```

Receive snapshot sensor data as friendly **JSON**:

```
"SNAPSHOTSENSOR": {
  "ResponseCode": { "Code": 0,"Desc": "OK" },
  "Temperature": 21.0
}
```

* JavaScript Object Notation



A tale of temperature (BLE)

There's a standard for temperature data, specified in **Section 3.8** of the **Assigned Numbers** document and **Section 3** of the **GATT Specification Supplement**:

3.8 Characteristics

Referenced from the following:

- Bluetooth Core Specification [Vol 3] Part C, Section 12 [4].
- Bluetooth Core Specification [Vol 3] Part F, Section 2 [4].

See Bluetooth Core Specification [Vol 3] Part G, Section 3.3 [4].

3.8.1 Characteristics by Name

Characteristic Name	UUID
Temperature	0x2A6E

https://www.bluetooth.com/specifications/assigned-numbers/

3.234 Temperature

The Temperature characteristic is used to represent a te

The structure of this characteristic is defined below.

Field	Data Type	Size (in octets)	Descri
Temperature	sint16	2	Base Unit: org.bluetooth.unit.thermodynamic_-temperature.degree_celsius Represented values: M = 1, d = -2, b = 0 Unit is degrees Celsius with a resolution of 0.01 degrees Celsius. Allowed range is: -273.15 to 327.67. A value of 0x8000 represents "value is not known". All other values are prohibited.

Table 3.360: Structure of the Temperature characteristic

https://www.bluetooth.com/specifications/gss/

21°C = **2100** x 0.01

0x0834 -> 0x3408

Big Endian -> Little Endian

The reality of temperature (BLE)

BLE affords much freedom… *…to reinvent the wheel, over and over.*

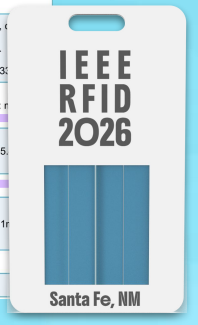
Byte offset	Field	Description
0	Frame Type	Value = `0x20`
1	Version	TLM version, value = `0x00`
2	VBATT[0]	Battery voltage, 1 mV/bit
3	VBATT[1]	
4	TEMP[0]	Beacon temperature
5	TEMP[1]	
6	ADV_CNT[0]	Advertising PDU count
7	ADV_CNT[1]	
8	ADV_CNT[2]	
9	ADV_CNT[3]	
10	SEC_CNT[0]	Time since power-on or reboot
11	SEC_CNT[1]	
12	SEC_CNT[2]	
13	SEC_CNT[3]	

Eddystone-TLM
Service: **0xfeaa**

Offset	Length	Type	Data	Details
0	1	Data Length	0x02	AD Structure Data Length
1	1	Flag Data Type	0x01	AD Type is Flags
2	1	Flag Data	0x06	LE General Discoverable Mode, BR/EDR not supported
3	1	Data Length	/	AD Structure Data Length
4	1	AD Type	0xFF	AD Type is Manufacturer Specific Data
5	2	Company ID	0x3906	LE, Manufacturer ID, 0x0639 = Minew
7	1	Frame Type	0xCA	Minew Connect V3
8	1	Frame Version	0x1B	Sensor Frame
9	1bit	Encrypto Flag	-	0: Disable, 1: Enable
9	7bit	Sub-version	0x01	Sub-version
10	2	Product ID	/	Product ID, 0x0007 = MTB02
12	1bit	CTE Flag	-	0: Disable, 1: Enable
12	5bit	CTE Length[1]	-	CTE Length Value, range 2~20, unit: 8us
12	2bit	Encrypto Range	-	Encryption Range, refer to Encrypto Range
13	6	MAC Address	/	LE, Product Serial Number ID, … equal to MAC address. 0x0165332211AC = AC:11:22:33
19	2	Battery	/	BE, Battery Voltage, Units: …
21	2	SOC Temperature	/	8.8 Fixed Point, 0x1973 => 25.
23	…	…	…	…
24	1	Measured Power	/	Calibrated RSSI value at 1…
25	2	Nonce Salt	/	-
27	4	Nonce Counter	/	-

Object id	Property	Data type	Factor	Example	Result	Unit
`0x57`	temperature	sint8 (1 byte)	1	`57EA`	-22	°C
`0x58`	temperature	sint8 (1 byte)	0.35	`58EA`	-7.7	°C
`0x45`	temperature	sint16 (2 bytes)	0.1	`451101`	27.3	°C
`0x02`	temperature	sint16 (2 bytes)	0.01	`02CA09`	25.06	°C

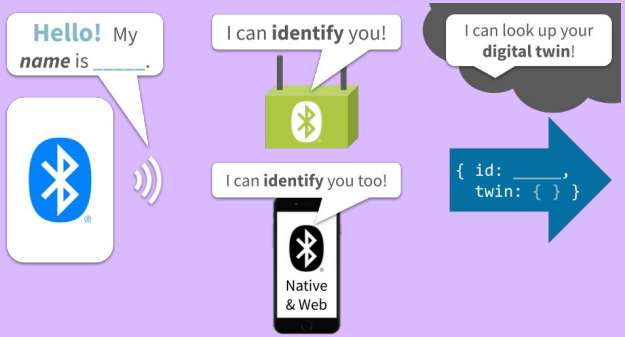
BTHome.io
Service: **0xfcd2**



Minew Connect
Company: **0x0639**

Best practices for BLE

For those who *don't* enjoy reading protocol specification documents.



Best Practices for BLE Identifiers

https://reelyactive.github.io/diy/best-practices-ble-identifiers/

Best Practices for BLE Advertising Data

https://reelyactive.github.io/diy/best-practices-ble-advertising-data/

advlib: advertising data library

https://github.com/reelyactive/advlib

😬 Raw binary

🤓 Friendly JSON

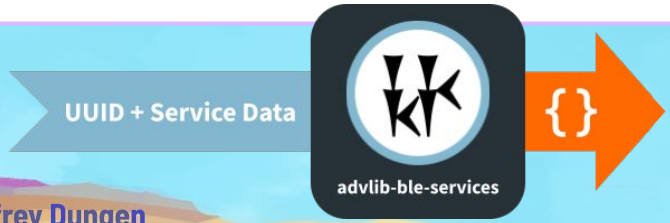
advlibs , not Mad Libs*

Eddystone-TLM data (BLE): 20000bb8150000000004500000258

github.com/google/eddystone/

Once upon a time, a weary beacon whose battery was only **3V** measured a temperature of **21C** it shouted for the **69th** time, exclaiming "did you know I've been up for **60** seconds?!?"

* Fun fact:
Mad Libs still exist!
madlibs.com



Dynamic ambient data

Common, vendor-and-technology-agnostic properties* include:

acceleration
ammoniaConcentration
amperage
angleOfRotation
angularVelocity
batteryPercentage
batteryVoltage
carbonDioxideConcentration
count
dissolvedOxygen
distance
duration
elevation
energy
heading
heartRate
illuminance
isButtonPressed
isCarbonMonoxideDetected
isContactDetected
isGasDetected
isHealthy
isInputDetected
isLightDetected
isLiquidDetected
isMotionDetected
isOccupancyDetected
isSmokeDetected
isTamperDetected
languages
levelPercentage
luminousFlux
magneticField
methaneConcentration
nearest
nitrogenDioxideConcentration
nitrogenOxidesIndex
numberOfOccupants
passageCounts
pH
pm1.0 / pm2.5 / pm10
position
power
pressure
relativeHumidity
soundPressure
speed
temperature
text
txCount
txCycle
uptime
velocityOverall
voltage

For the **TPMS reports** in Kevin's tutorial 😉

* defined & observed by reelyActive's middleware

https://reelyactive.github.io/diy/cheatsheet/#dynamb

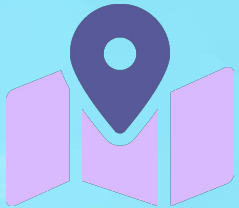
Decoding metadata

A radio decoding may result in more than just **RSSI**...

- ➔ **AoA** (angle of arrival)
- ➔ **TDOA** (time difference of arrival)
- ➔ **Range** (ex: phase-based ranging, round-trip time, ToF, ...)
- ➔ **IQ samples** (ex: to calculate AoA) [https://wikipedia.org/wiki/In-phase_and_quadrature_components]
- ➔ **Frequency**
- ➔ **Phase**

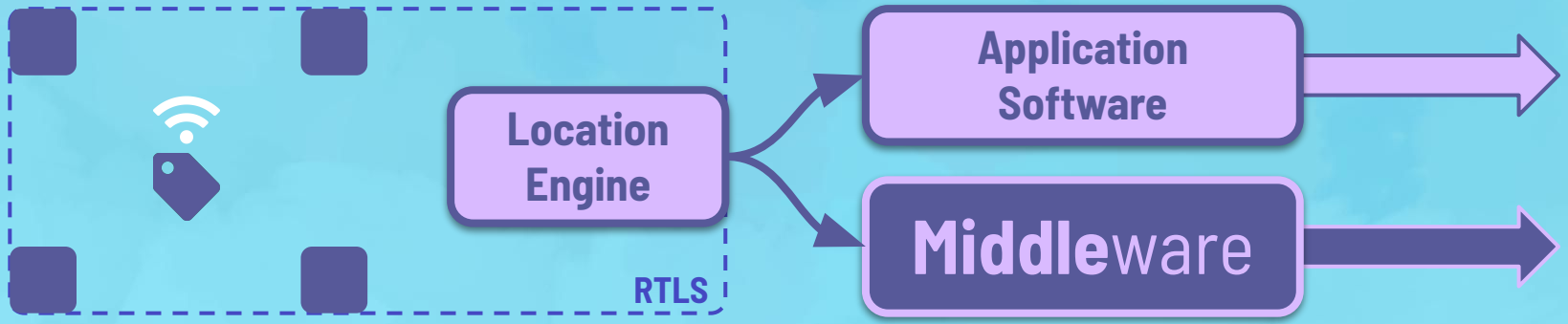
Recall Dan's tutorial!

Often, these are used for absolute or relative **positioning** estimation.



Real time location systems (RTLS)

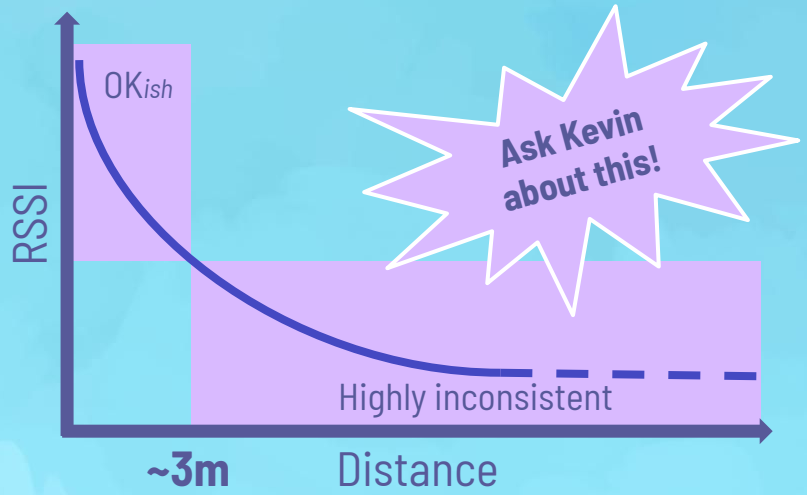
A RTLS combines hardware and a **location engine** to estimate the position of tags/beacons based on decoding metadata.



Some RTLS vendors bundle application software to create an end-to-end solution. Middleware may also be used.

At least we always have RSSI

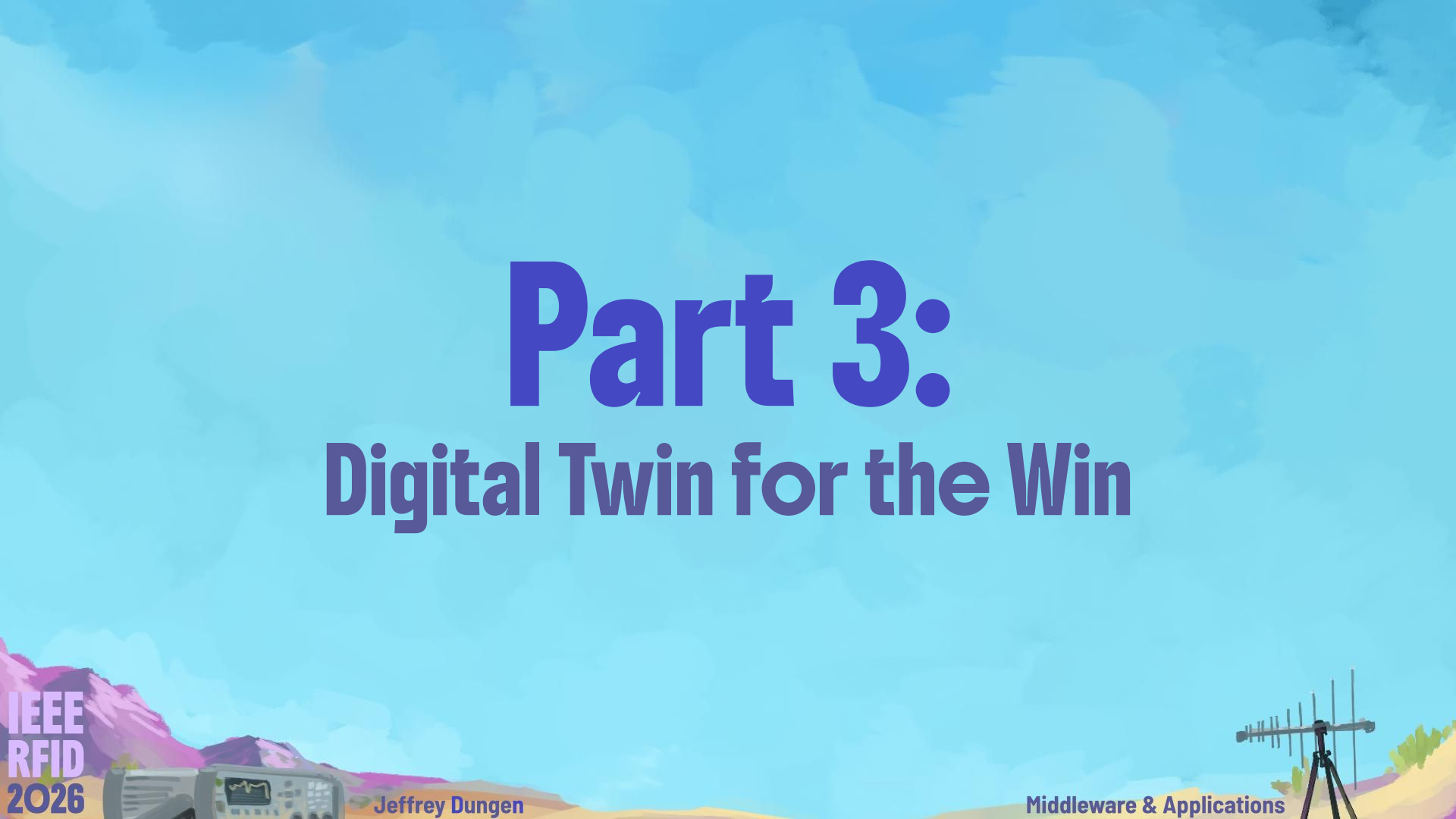
Simple, ubiquitous **RSSI-based** position estimation may nonetheless be "good enough" provided sufficient infrastructure density*.



* Does it make more sense to:
- ➔ install a gateway/antenna every 3m (10ft)?
- ➔ invest in a complex RTLS?

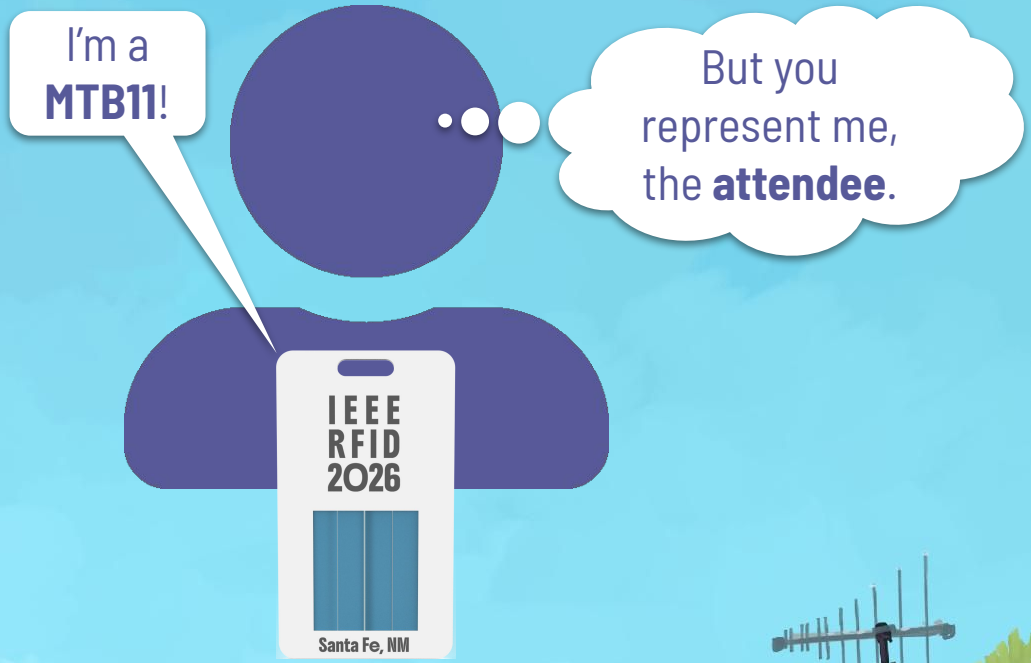
Write your own positioning/location engine:
https://github.com/reelyactive/chimps

Part 3:
Digital Twin for the Win



A digital twin of what?

Are we “attached” to a purist definition of a digital twin? 🤔

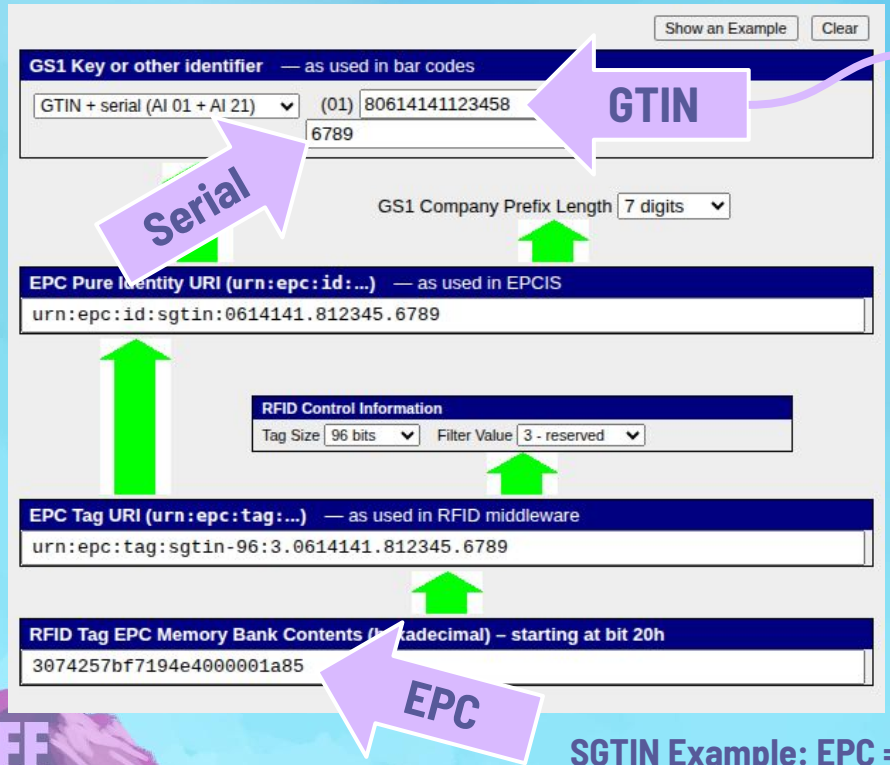


Advertising a URL (BLE)

Bluetooth Low Energy devices can *spontaneously* broadcast a (short*) **URL** (in a GAP-standard way!) to any observers in range.

* The URL above is just too long to fit in a *legacy* BLE advertising packet.

Product lookup (RAIN & GS1)



https://www.gs1.org/services/epc-encoderdecoder
https://www.gs1.org/services/verified-by-gs1/results?gtin=80614141123458

SGTIN Example: EPC = SGTIN-96
Serialised Global Trade Item Number

3074257bf7194e4000001a85

EU Digital Product Passport

The European Union DPP, which is currently in a phased roll-out, will be *mandatory* for certain products sold in the EU.

- ➔ A DPP is linked to a product by an **identifier**
- ➔ The identifier can be in the form of a QR code, **RFID tag**, etc.
- ➔ DPP **information** may be publicly available, or user-privileged
- ➔ A DPP may allow **updates** as it follows the product life cycle

The emerging standard(s) should serve as a blueprint for broader **digital twin initiatives** worldwide.

https://untp.unece.org/
https://therainalliance.org/gs1-and-rain-alliance-joint-dpp-statement/

Digital twin applications

RFID, *and more generally Auto-ID**, enable machine-readable links between **physical things** and their **digital representations**.

We've also seen that a real-time location system (RTLS) provides a digital representation of their **physical location**. And we've seen that ambient **sensor data** can be digitally represented too.

🤔 *Can middleware mix all these together into something useful?*

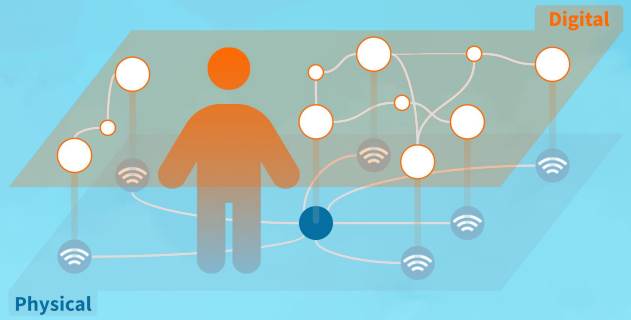
* 50+ years of barcodes!

Hyperlocal Context

"A machine-readable, real-time contextual representation of a physical space and its occupants."

Hyperlocal context is a **digital snapshot** of *who/what* is *where/how*, achieved by combining:

- ➔ Real-time location
- ➔ Digital twins
- ➔ Live sensor data



https://www.reelyactive.com/context/

Hyperlocal Context tutorial(s)

Once again, there's an IEEE RFID tutorial for that!*

Time for a live demo?

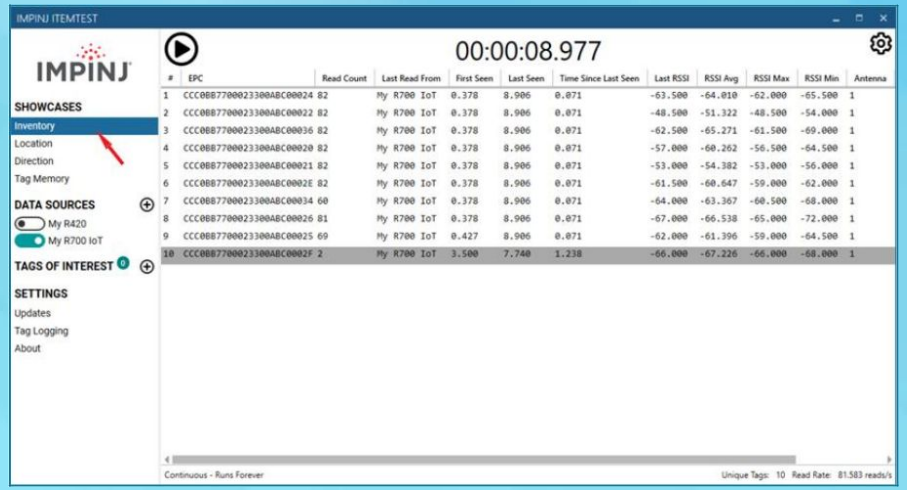
https://www.reelyactive.com/science/ieeerfid/

* Actually, *two* tutorials for that!

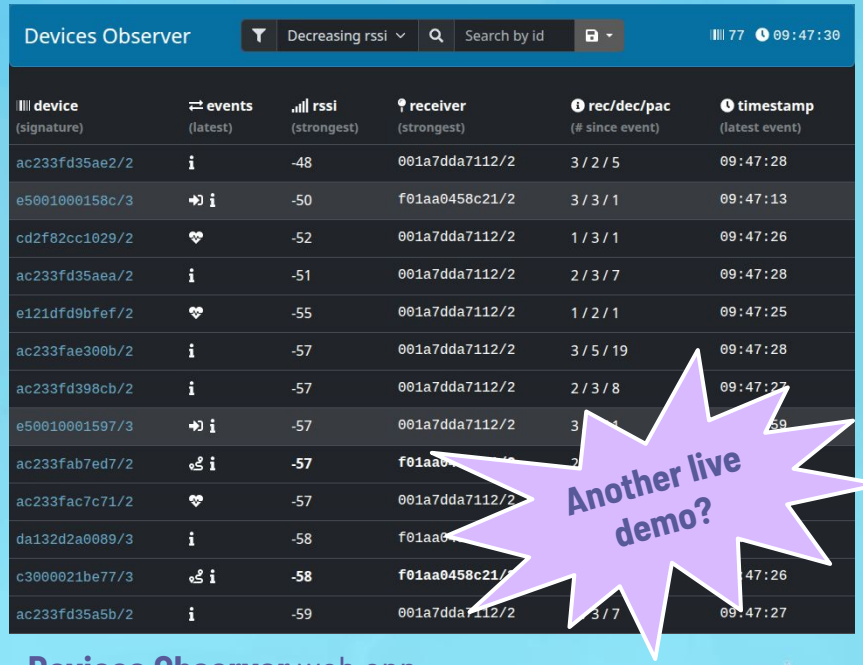
Part 4:

Application of every*thing*

Real-time data



Impinj ItemTest for Windows (Inventory Showcase)
Source: ItemTest User Manual



Devices Observer web app
Demo: https://reelyactive.github.io/pareto-anywhere-apps/devices-observer/?demo=default

A quick note on transport

There are many ways to transport data between the “wares”:

- ➔ **MQTT**: publish/subscribe messaging protocol
- ➔ **WebSocket**: bidirectional real-time comms over TCP
- ➔ **Webhook**: HTTP POST to endpoint
- ➔ **TCP / UDP**: low-level connection-oriented/less protocols

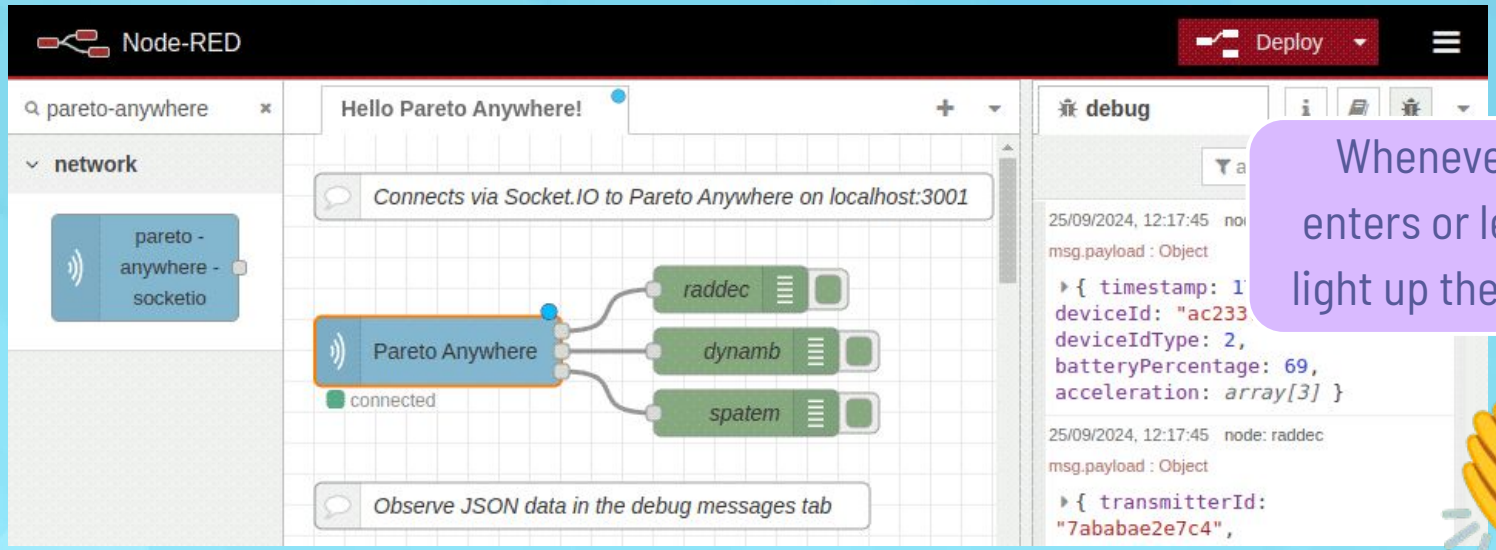
Industrial standards that combine transport & structured data:

- ➔ **Sparkplug**: MQTT with Protobufs, payload/state definitions, …
- ➔ **OPC-UA**: transport agnostic communication/info standard*

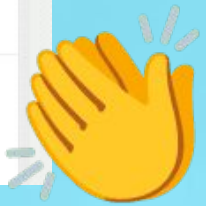
* standardises telemetry and, now, “unified locating” for spatial intelligence too!

Event-driven applications

Flow-based, low-code, event-driven programming.



Whenever Jeff's badge enters or leaves the stage, light up the "Applause" sign.



Node-RED open source software

https://nodered.org/ See also: https://flowfuse.com/

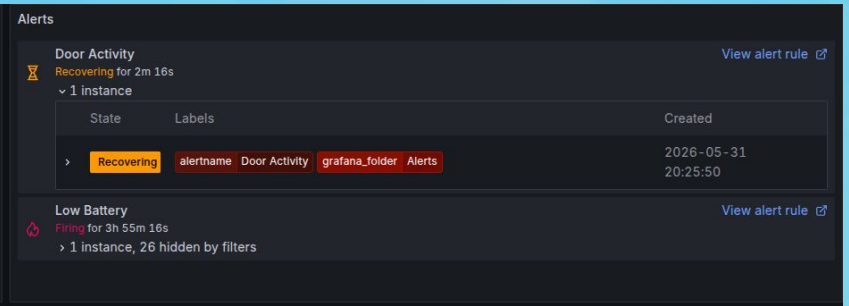
Databases

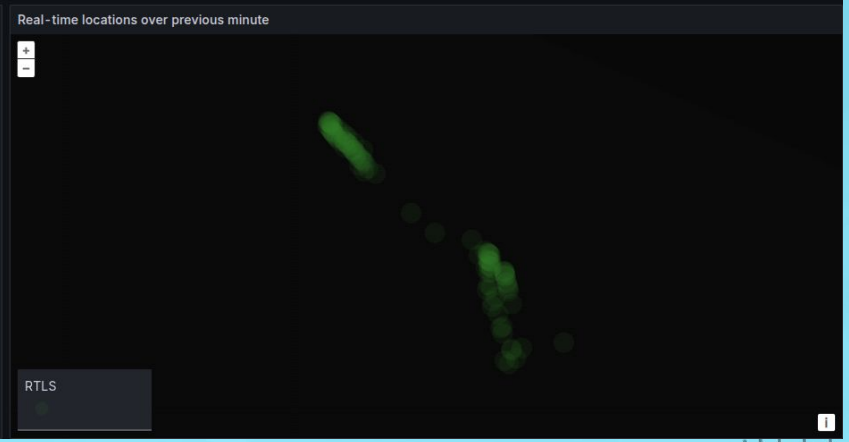
Store and query historical data. Flavours include:

- **SQL**: relational databases that support SQL queries
- **NoSQL**: non-relational databases that don't require data to conform to a schema
- **Time Series**: databases optimised for compressing and querying high volumes of timestamped data

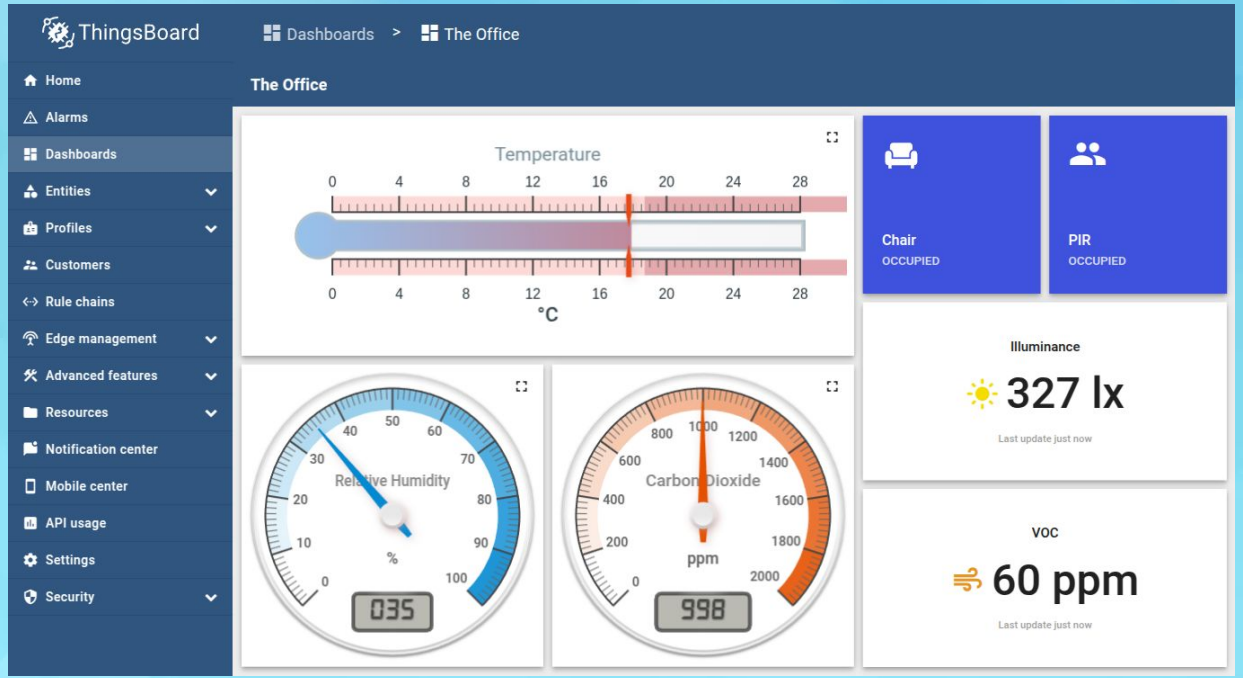
Middleware generally supports a variety of common databases, ideally the database you already use!

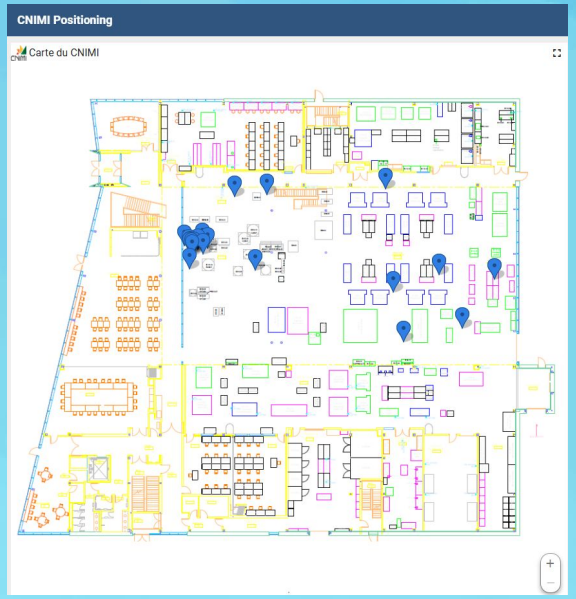
Dashboards (ex: Grafana)





Suites (ex: ThingsBoard)



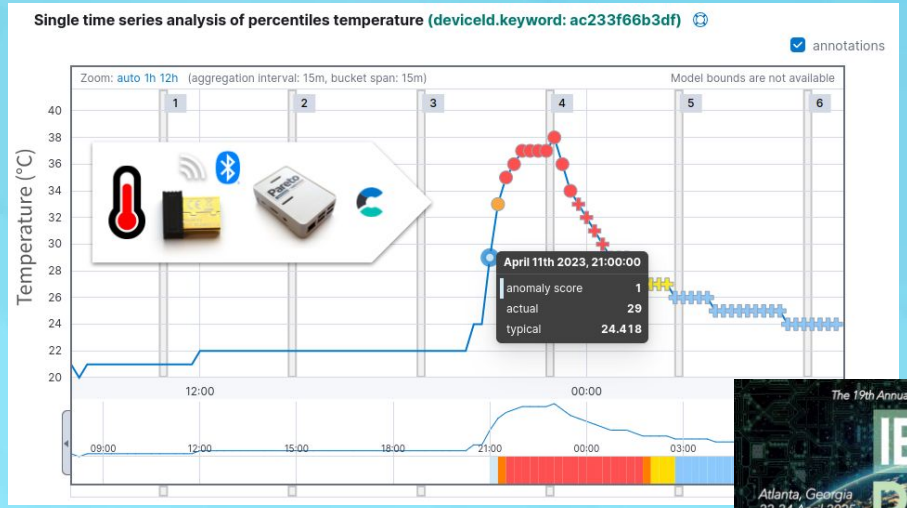


Real-time location of RAIN RFID tags at CNIMI, in Drummondville, Québec.

Suite combines database, dashboard, flows, etc.

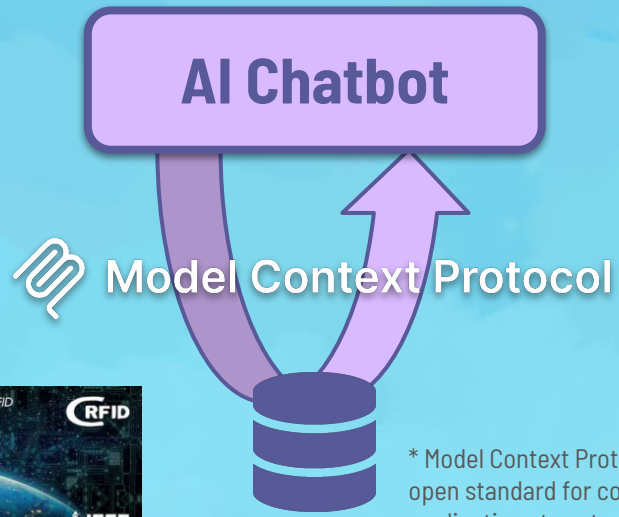
Artificial Intelligence

Anomaly Detection of time series data



Real example using Elastic Stack

MCP* interface to databases

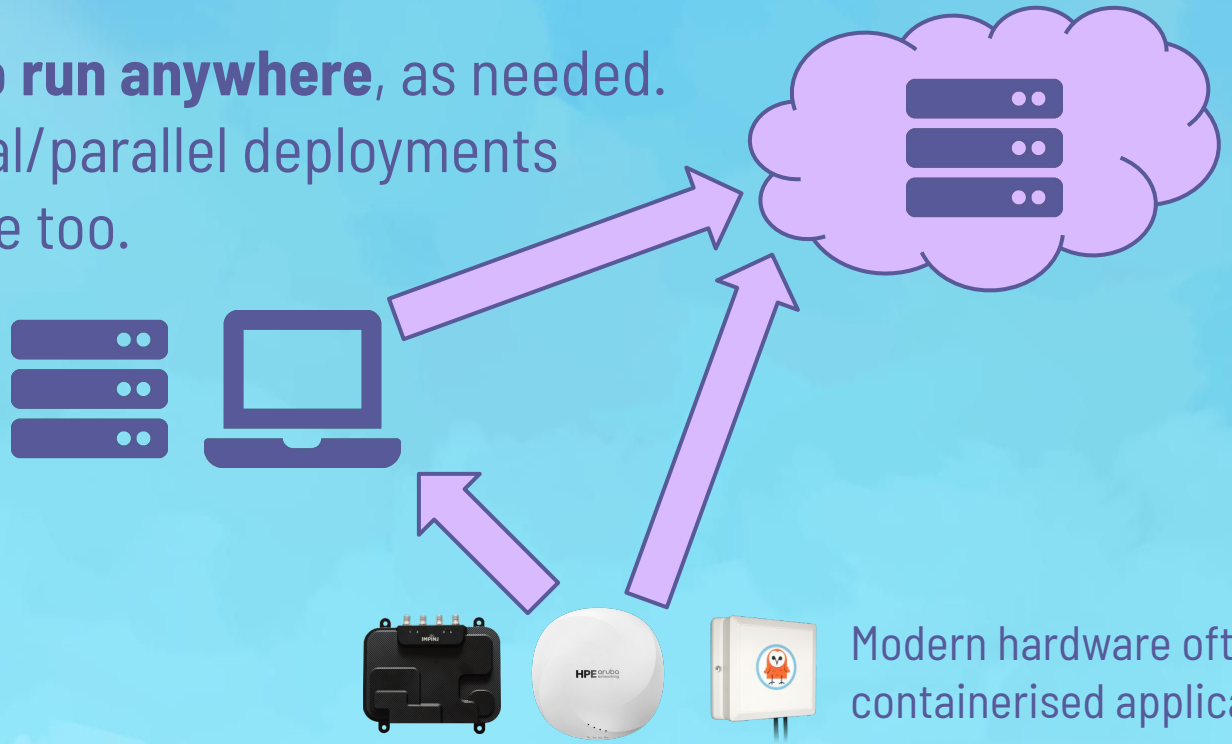


* Model Context Protocol is an open standard for connecting AI applications to external systems.

https://www.reelyactive.com/science/ieeerfid/reelyActive-RFIDinAIoT-250422.pdf

Where do middleware & apps run?

Common to **run anywhere**, as needed.
Hierarchical/parallel deployments are possible too.



Modern hardware often supports containerised applications.

Summary

Hardware does the hard part, but leaves a *mess* of data/interfaces.

Middleware tidies the mess, making data *usable* for software.

Software enables *applications*, Kevin* will present examples next.

Middleware & Applications

IEEE RFID 2026 Tutorial

presented by

Jeffrey Dungen

Up next!