

RIGEL

Real-World Interactive Games and Electronics Link

Windows Application and RIGEL-WEB Communication Protocol

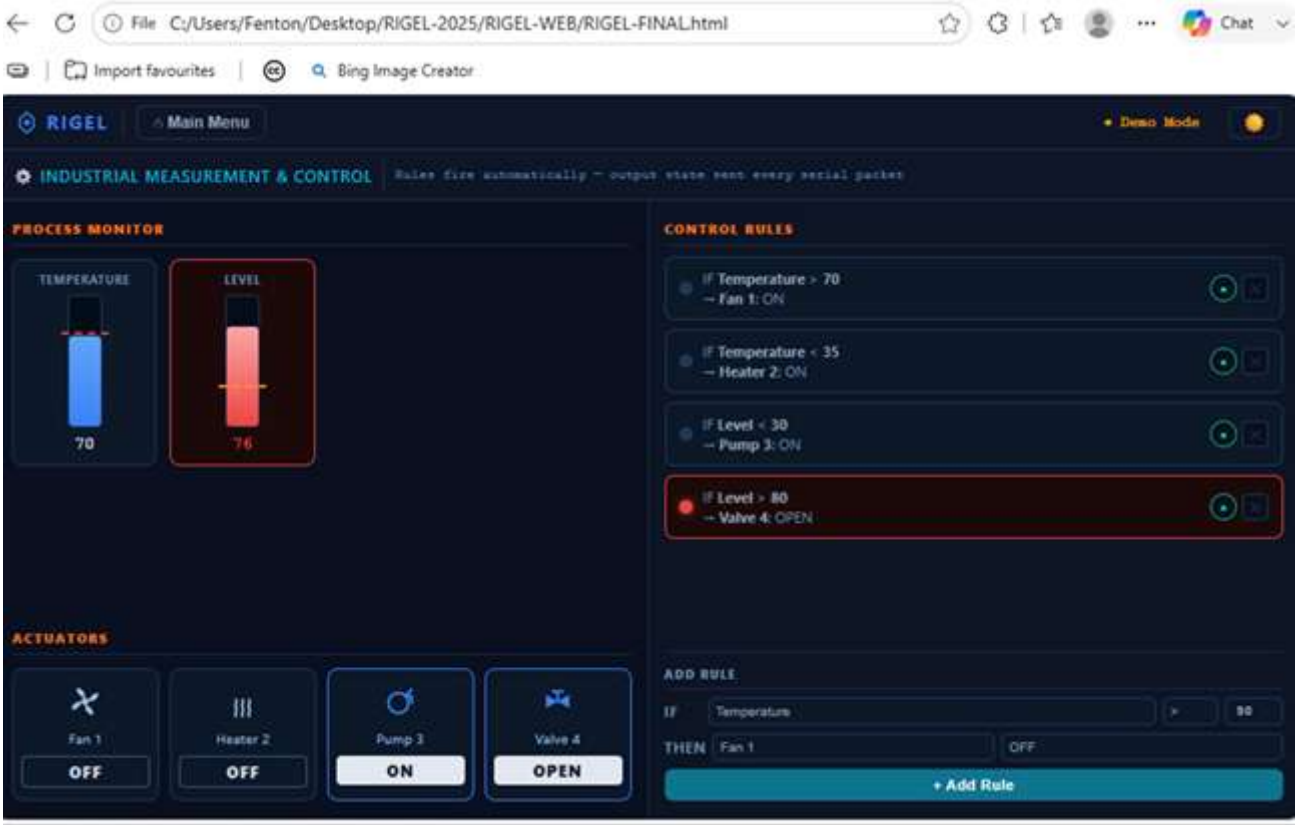
Technical Reference — Picaxe, ESP/Arduino, and BBC Micro:bit to RIGEL or RIGEL-WEB

Author: Michael Fenton MRSNZ

Related Zenodo records:

- Authentic Learning Using Mobile Sensor Technology (2008): https://doi.org/10.5281/zenodo.19302276
- RIGEL — Learning From Life, Kuala Lumpur (2009): https://doi.org/10.5281/zenodo.19334228
- Casio Calculator Data Logger Technical Reference (2025): https://doi.org/10.5281/zenodo.19281514
- RIGEL - Real-World Interactive Games and Electronics Link: A Universal Sensor Interface Invented by Subverting Game Design Software for Authentic STEM Learning (2026). https://doi.org/10.5281/zenodo.19508521
- RIGEL-WEB: The Real-World Interactive Games and Electronics Link as a Self-Contained HTML Sensor Interface Invented by Subverting Web Browser Architecture for Authentic STEM Learning (2026): https://doi.org/10.5281/zenodo.20541487

https://mikefentonnz.github.io/



Introduction

RIGEL System Overview

The Real-world Interactive Games and Electronics Link (RIIGEL) system was developed in response to an ongoing problem in STEM education: how do you engage learners in authentic hands-on science and mathematics investigations with no equipment and no budget?

The solution was to identify structural properties of existing tools and exploit them for purposes the designers did not intend — then document the process and result for others to replicate.

In 2008, the first classroom-verified research report demonstrated two such solutions using design by subversion:

- A Casio FX-9750 calculator was used as a robot remote control and data logger.
- A drag-and-drop game engine (GameMaker) became a universal graphical science instrument — at no cost.

This document covers the second of these: the communication protocol between a microcontroller and the Windows RIGEL GameMaker application and the 2026 RIGEL-WEB HTML port as a standalone browser app.

Two Operational Modes — One Firmware Platform

The RIGEL microcontroller firmware supports two distinct operational modes. The microcontroller connects to ONE device at a time — either a Casio FX-9750 calculator or a Windows PC/laptop running the RIGEL GameMaker application. Both modes are never active simultaneously.

Mode	Device Connected	Communication	Mobility
Casio Mode	Casio FX-9750 calculator	3-pin serial (SB-62 cable)	Fully portable — handheld calculator
RIGEL Mode	Windows PC / laptop	RS232 serial or Bluetooth	Mobile when using Bluetooth

The Bluetooth option is significant: a student wearing a biomedical sensor connected to a Picaxe or ESP32 transmitting via Bluetooth to a laptop running RIGEL can move freely through an environment while streaming real-time data — heart rate during physical activity, motion during sport, or environmental readings during fieldwork.

Historical Context

Original Work (2004–2008)

Michael Fenton pioneered the use of game design software as a science instrument from 2004 onward, first documented in:

- Fenton, M. (2008). Authentic Learning Using Mobile Sensor Technology With Reflections on the State of Science Education in New Zealand. Ministry of Education E-Learning Fellowship report. https://doi.org/10.5281/zenodo.19302276

RIGEL Presented Internationally (2009)

- Fenton, M. (2009). RIGEL — Learning From Life — Communities of Learning via a Connected Curriculum. Microsoft Partners in Learning Regional Innovative Teachers Conference, Kuala Lumpur. https://doi.org/10.5281/zenodo.19334228

RIGEL was recognised with the Microsoft New Zealand Innovative Teacher Award. Michael Fenton represented New Zealand at the Asia Pacific Microsoft Innovative Teachers Conference in Kuala Lumpur, 2009.

GameMaker Version History

RIGEL was originally written in GameMaker 4 by Mark Overmars, which was freely licensed for non-commercial use. The current version uses GameMaker 6.

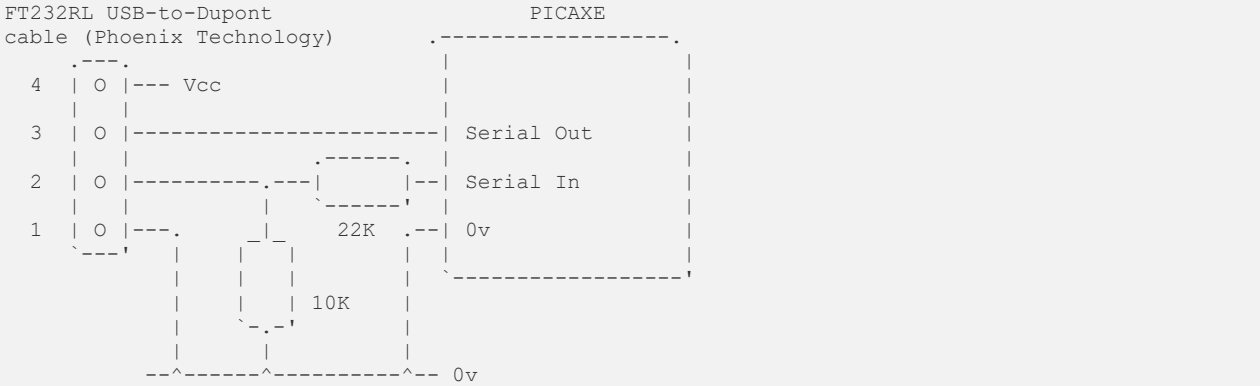
- GameMaker 4 original licence: https://creatools.gameclassification.com/EN/creatools/49-Game-Maker-4.0/index.html
- GameMaker 6 (archived): https://archive.org/details/gamemaker61a

The game design software interprets incoming serial data as keyboard or mouse events. Instead of a player character responding to keyboard input, dials, gauges, graphs, and data displays respond to real-world sensor values. The game engine becomes a science instrument.

Hardware Interface

USB-to-Serial Connection (Wired) for Picaxe Microcontrollers

The FT232RL USB-to-Dupont cable (Phoenix Technology) provides RS232 serial communication between the RIGEL GameMaker application and the Picaxe microcontroller.



NOTE: Other microcontrollers (e.g., ESP32 or Microbit) can connect using standard USB data cable.

Serial Parameters

Parameter	Value	Notes
Baud rate — Picaxe 08M2 / 14M2	19200	Set frequency to 16MHz; use SERIN / SEROUT (not HSEROUT)
Baud rate — ESP8266 / ESP32	38400 max	Maximum when using USB as RS232 terminal connection
RIGEL room speed	40	Main menu requests data at room_speed/4 = 10Hz
Effective data rate	4 Hz (4 times/second)	Sufficient for CPR compression detection

Bluetooth (Wireless) Connection

For wireless operation, a Bluetooth serial module is connected to the microcontroller's serial pins in place of the USB-RS232 cable. The RIGEL GameMaker application connects to the Bluetooth serial port as a standard COM port. This enables:

- Wearable biomedical sensor data streaming (heart rate, motion, respiration)
- Mobile environmental monitoring during fieldwork
- Game control using body movement or exercise equipment (e.g., exercise bike as flight simulator controller)
- Remote robot sensing and control

Communication Protocol

Overview — Request/Response Handshaking

The RIGEL communication protocol uses a request/response model. The GameMaker application always initiates communication by sending a handshaking byte. The microcontroller responds only when requested — it never transmits unsolicited data. This prevents data collisions and ensures synchronisation.

Direction	Content	Purpose
GameMaker → Microcontroller	'R' (DEC 82) + optional output device states	Request sensor data; optionally set output device states
Microcontroller → GameMaker	'R' + packet_length + sensor data + output states + \\n	Send sensor readings and confirm output device states

Key rules:

- GameMaker ALWAYS sends 'R' to request data and MAY include output device state changes.
- GameMaker NEVER sends sensor data back — this would increase packet size unnecessarily.
- The microcontroller changes output device states first, then responds with updated sensor and output state data.
- If a byte value after packet_length is > 90 it is a sensor type identifier. If < 90 it is an output device state.

Packet Structure

Start Byte and Packet Length

Every packet begins with the START byte 'R' (ASCII DEC 82) followed by a single byte giving the total length of the data payload:

Byte position	Content	Value	Description
1	START_byte	'R' (DEC 82)	Handshaking identifier — always 'R'
2	packet_size	Integer	2 × sensor_count + 1 × output_count
3 onward	Data bytes	Varies	Sensor pairs and/or output device states
Last	END_byte	\\n (DEC 10)	Newline — marks end of packet

Packet size formula:

```
packet_size = (2 × sensor_count) + (1 × output_count)
```


Sensor Data Encoding

Each sensor occupies TWO bytes in the packet:

- Byte 1 (sensor type): ASCII code for the sensor type letter (e.g., 't' = 116 for temperature). All sensor type codes are > 90, distinguishing them from output device codes.
- Byte 2 (sensor value): An integer from 0–100 representing the sensor reading, with special encodings for temperature, pressure, compass heading, and distance.

Special value encodings:

Sensor	Encoding	Example
Temperature	0–100 = 0°C to 100°C; >100 = negative (105 = -5°C)	20°C → send 20; -5°C → send 105
Barometric pressure	Send (reading − 900) to fit within 0–255	1013 hPa → send 113
Compass heading	1–16 representing N, NNE, NE, ENE, E, ESE, SE, SSE, S, SSW, SW, WSW, W, WNW, NW, NNW	North → send 1; South → send 9
Distance (HC-SR04)	Raw cm value, maximum 254 (2.54 metres)	50 cm → send 50
All other sensors	Raw percentage 0–100	75% → send 75

Output Device Encoding

Output device states each occupy ONE byte. The byte is interpreted using modulo 10 arithmetic:

- The TENS digit identifies the device type.
- The ONES digit identifies the device state.

Tens digit	Device type	Example byte
1x	Switch	10 = off, 11 = on
2x	Motor	20 = off, 22 = forward, 23 = backward, 25 = robot off, 26 = robot forward, 27 = robot backward, 28 = robot left, 25 = robot right
3x	LED / RGB LED	34 = red, 35 = green, 36 = blue, 37 = yellow, 38 = white
4x	Fan	40 = off, 41 = on
5x	Heater	50 = off, 51 = on
6x	Vent	60 = closed, 61 = open
7x	Pump	70 = off, 71 = on
8x	Valve	80 = closed, 81 = open

Types 4–8 are binary actuators added for the Industrial Measurement & Control module (and usable anywhere, e.g. a robot fitted with a fan or heater). The firmware declares the device kind directly in the tens digit, so the panels need no role configuration — the decoder still just splits tens = type, ones = state. State 1 reads as **ON** for fan/heater/pump/switch and as **OPEN** for vent/valve; state 0 is OFF / CLOSED.

Distinguishing Sensors from Output Devices

The receiving device determines whether each byte is a sensor type or an output device state by its value:

```
IF byte_value > 90 THEN it is a sensor type byte → read the next byte as the sensor value

IF byte_value < 90 THEN it is an output device state → decode using Mod 10
```

This is reliable because all sensor type ASCII codes start at 97 ('a'), while all output device state codes are below 90. Eight output device types is therefore the ceiling; do not add a type 9 without first redesigning the 90 boundary.

Motor profiles and turn codes:

Auto-detects, from the first packet, whether a motor device should be controlled as a plain single motor or as a two-motor robot drive. The two profiles use disjoint state ranges, so the first packet a device reports tells the PC which control surface to render — no manual configuration and no extra protocol field.

A robot drive is interpreted by the firmware as two motors: turn left runs one motor forward and the other reverse (and vice-versa for turn right), while forward and reverse drive both motors the same way. A single motor uses the original forward and reverse codes.

Ones digit	State
0	Off
2	Single motor Forward
3	Single motor Backward
5	Robot drive Off
6	Robot drive Forward (both motors forward)
7	Robot drive Backward (both motors reverse)
8	Robot drive Turn left (motors opposed)
9	Robot drive Turn right (motors opposed)

1 and 4 are reserved (24 is a deliberate buffer between the two ranges). A single motor only ever emits states 0–3; a robot drive only ever emits 5–9. The firmware contract is that each motor reports its idle/startup state inside its own range — a robot drive idles at 25, not 20 — so the first-packet classification is unambiguous. All motor states latch: the microcontroller holds each output until the PC sends a change. Turn left and turn right appear only on the drive profile.

Sensor Encoding Reference

All sensor type identifiers are assigned as ASCII byte values of lowercase letters. The Picaxe SYMBOL declarations below are used in firmware code for readability. RIGEL automatically assigns units of measurement based on the sensor type received.

Symbol	ASCII	Sensor type	Value encoding
a_	97	Atmospheric (barometric) pressure	Send (hPa reading − 900); ADD 900 to recover original value
b_	98	Blue colour sensor	Raw ANALOG 0–100; use alarm min/max for digital threshold
c_	99	Contact / impact / break beam	DIGITAL: 0 = no contact, 1 = contact
d_	100	Distance sensor (HC-SR04)	Raw cm, maximum 254 (2.54 m)
e_	101	Compass heading	1–16: N, NNE, NE, ENE, E, ESE, SE, SSE, S, SSW, SW, WSW, W, WNW, NW, NNW
g_	103	Green colour sensor	Raw ANALOG 0–100
h_	104	Humidity	Raw percentage 0–100
i_	105	Infrared (IR)	Raw ANALOG 0–100
l_	108	Ambient light (LDR)	Raw ANALOG 0–100
m_	109	Magnetic field strength	Raw ANALOG 0–100
n_	110	Nuclear particle (alpha)	Raw ANALOG 0–100
o_	111	O2 / respiration	Raw ANALOG 0–100
p_	112	Pulse / blood flow	Raw ANALOG 0–100
r_	114	Red colour sensor	Raw ANALOG 0–100
s_	115	Sound level	Raw ANALOG 0–100
t_	116	Temperature (°C)	0–100 = positive; >100 = negative (105 = −5°C)
u_	117	UV light	Raw ANALOG 0–100
x_	120	START timer	Timer control command
y_	121	Yellow colour sensor	Raw ANALOG 0–100
z_	122	STOP timer	Timer control command

Complete Packet Examples

Example 1: Single Temperature Sensor, No Output Devices

Microcontroller has one sensor: temperature reading of 22°C. No output devices.

Byte	Value (DEC)	Value (ASCII/HEX)	Meaning
1	82	'R' (0x52)	START — handshake
2	2	0x02	Packet length = 2 (1 sensor × 2 bytes)
3	116	't' (0x74)	Sensor type: temperature
4	22	0x16	Temperature reading: 22°C
5	10	\\n (0x0A)	END byte

Example 2: Three Sensors — Temperature, Light, Sound

Temperature = 20°C, light = 65%, sound = 40%. No output devices.

Byte	Value (DEC)	Meaning
1	82	START 'R'
2	6	Packet length = 6 (3 sensors × 2 bytes)
3	116	Sensor type: temperature ('t')
4	20	Temperature: 20°C
5	108	Sensor type: light ('l')
6	65	Light: 65%
7	115	Sensor type: sound ('s')
8	40	Sound: 40%
9	10	END \\n

Example 3: Two Sensors and One Output Device (LED)

Temperature = 30°C, humidity = 55%. LED is currently green (35).

Byte	Value (DEC)	Meaning
1	82	START 'R'
2	5	Packet length = 5 (2 sensors × 2 bytes + 1 output × 1 byte)
3	116	Sensor type: temperature ('t')
4	30	Temperature: 30°C
5	104	Sensor type: humidity ('h')
6	55	Humidity: 55%
7	35	Output device: LED green (tens=3 → LED; ones=5 → green)
8	10	END \\n

Example 4: Negative Temperature (−5°C)

Temperature values above 100 signify negative temperatures. Send (100 + |value|).

Byte	Value (DEC)	Meaning
1	82	START 'R'
2	2	Packet length = 2
3	116	Sensor type: temperature ('t')
4	105	Temperature: 105 → decoded as −5°C
5	10	END \\n

Example 5: Barometric Pressure (1013 hPa)

Send (hPa − 900) to fit within the 0–255 byte range. RIGEL adds 900 to recover the original value.

Byte	Value (DEC)	Meaning
1	82	START 'R'
2	2	Packet length = 2
3	97	Sensor type: atmospheric pressure ('a')
4	113	113 + 900 = 1013 hPa
5	10	END \\n

RIGEL GameMaker Application

MS-Windows Application Architecture

RIGEL is written in GameMaker 6 using drag-and-drop block coding and the GameMaker Language (GML). The central architectural innovation is the substitution of serial sensor data for keyboard and mouse events: the game engine responds to real-world physical measurements exactly as it would respond to player input — causing dials, graphs, data displays, and game characters to respond to sensor values.

Different rooms (levels) within RIGEL serve as different instrument panels or display environments:

- Science lab sensor dashboard — real-time dials, meters, and graphs
- 3D flight simulator — controlled by an exercise bike motion sensor
- CPR training simulator — pressure sensor in a manikin provides real-time compression feedback
- Home security monitoring centre
- Industrial measurement and control (IMC) system simulation
- Remote robot sensing and control
- Universal game controller using body-worn or DIY sensors

Timing and Data Rate

RIGEL room speed is set to 40 steps per second. The Main Menu requests data at room_speed/4, giving an effective data acquisition rate of 4 Hz (4 requests per second). This is sufficient for:

- Real-time environmental sensor monitoring
- CPR compression detection (compressions faster than once per second are detected reliably)
- Biomedical data streaming (heart rate, respiration, motion)
- Game control using physical sensors

Auto-Detection of Connected Hardware

RIGEL auto-connects and recognises sensors and output devices on Picaxe, Arduino, ESP8266, and ESP32 development boards. When RIGEL receives a data packet, it reads the sensor type byte and automatically:

- Identifies the sensor type from the ASCII letter code
- Assigns the correct unit of measurement to the display
- Routes the value to the appropriate gauge, graph, or game element

This means students do not need to configure RIGEL when changing sensors — the firmware sends the sensor type letter and RIGEL adapts automatically.

Sound and Background Music

Note for developers: when using background music in RIGEL with the sound_loop() function, audio files MUST be imported as WAV format. This prevents the background music stopping when a sound effect (e.g., an explosion or alert) plays simultaneously. MP3 files do not support concurrent playback in GameMaker 6.

RIGEL-WEB HTML Web Broswer Application

RIGEL-WEB has similar features listed above – auto detection, demonstration mode, science lab module, IMC module, home security module, etc.

Microcontroller Compatibility

RIGEL and REIGEL-WEB work with the following microcontroller platforms using the same serial protocol:

Microcontroller	Baud rate	Serial type	Notes
Picaxe 08M2	19200	SERIN / SEROUT	Set frequency to 16MHz; do NOT use HSEROUT
Picaxe 14M2	19200	SERIN / SEROUT	Set frequency to 16MHz; do NOT use HSEROUT
ESP8266	Up to 38400	USB serial / Bluetooth	Use USB as RS232 terminal connection
ESP32	Up to 38400	USB serial / Bluetooth	Use USB as RS232 terminal connection
BBC Micro:bit	Varies	Serial / Bluetooth	Compatible via serial passthrough
Arduino (compatible)	Up to 38400	USB serial / Bluetooth	Standard Arduino serial library

All devices use the same packet structure and encoding. The protocol is designed to be hardware-agnostic: changing the microcontroller platform requires no changes to the RIGEL GameMaker application.

Troubleshooting

Symptom	Likely cause	Solution
No data received by RIGEL	Wrong COM port selected	Check Device Manager for correct COM port number
No data received by RIGEL	Wrong baud rate	Picaxe: set 19200; ESP32: set 38400 or lower
Data received but values wrong	Sensor type byte incorrect	Verify sensor symbol code matches sensor encoding table
Output device not responding	Incorrect Mod 10 encoding	Verify tens digit = device type, ones digit = state
Background music stops on sound effects	MP3 file format used	Re-import music as WAV — WAV files support concurrent playback
Intermittent data loss	Baud rate too high for Bluetooth	Reduce baud rate; Bluetooth latency may require lower rate than wired
Packet misalignment	Missing END byte	Ensure microcontroller sends \\n (DEC 10) after every packet

Educational Applications

Why Use RIGEL or RIGEL-WEB for Data Logging or Games/Instrumentation?

Advantage	Detail
Zero software cost	GameMaker 4/6 was free for non-commercial use; archived versions available
Universal interface	Same application works with any sensor on any supported microcontroller
Gamified learning	Science instruments presented as game interfaces increase engagement
Wearable capability	Biomedical sensors via Bluetooth enable body-worn data collection
Cross-curricula	Integrates science, mathematics, coding, art, music, games, and physical education
Student-built sensors	DIY sensors costing cents replace commercial sensors costing hundreds of dollars
Scalable	Works from primary school through to postgraduate level without modification

Suitable Investigations

Physical Science

- Temperature vs time (Newton's cooling, heat transfer)
- Light intensity (inverse square law, Beer's law, colourimetry)
- Sound level surveys (noise mapping, sound proofing tests)
- Alpha particle detection (ionising radiation, background counts)
- Magnetic field mapping
- UV exposure monitoring

Biology and Health

- Heart rate monitoring (exercise recovery curves, resting rate comparisons)
- Respiration rate measurement
- CPR compression timing (gamified training simulator)
- Bioluminescence measurement

Environmental Science

- Atmospheric pressure logging
- Humidity and temperature mapping across environments
- Sound level monitoring in classrooms and outdoor spaces
- Transect sampling during fieldwork

Physical Computing and Game Design

- Exercise bike as 3D flight simulator controller
- Body motion as game character controller
- DIY game controllers using resistance or light sensors
- Home security monitoring system simulation
- Industrial measurement and control (IMC) system simulation

Industrial Measurement and Control

- Write rules that control real-world devices (fans, pumps, valves, etc) in response to sensor inputs.

References

- Fenton, M. (2008). Authentic Learning Using Mobile Sensor Technology With Reflections on the State of Science Education in New Zealand. Ministry of Education E-Learning Fellowship report. https://doi.org/10.5281/zenodo.19302276
- Fenton, M. (2009). RIGEL — Learning From Life — Communities of Learning via a Connected Curriculum. Microsoft Partners in Learning Regional Innovative Teachers Conference, Kuala Lumpur, Malaysia. https://doi.org/10.5281/zenodo.19334228
- Fenton, M. (2025). Rediscovering the 3-Pin Port: A Novel RECEIVE() Protocol Discovery Enabling Low-Cost Classroom Data Logging on Casio FX-9750 and FX-9860 Series Calculators. Zenodo. https://doi.org/10.5281/zenodo.19281514
- Fenton, M. (2026). RIGEL - Real-World Interactive Games and Electronics Link: A Universal Sensor Interface Invented by Subverting Game Design Software for Authentic STEM Learning. https://doi.org/10.5281/zenodo.19508521
- Fenton, M. (2026). RIGEL-WEB: The Real-World Interactive Games and Electronics Link as a Self-Contained HTML Sensor Interface Invented by Subverting Web Browser Architecture for Authentic STEM Learning (2026): https://doi.org/10.5281/zenodo.20541487
- Overmars, M. GameMaker 4 (archived). https://creatools.gameclassification.com/EN/creatools/49-Game-Maker-4.0/index.html
- GameMaker 6.1 (archived, with modern Windows patch). https://archive.org/details/gamemaker61a
- GameMaker Engineering Archive (GM project disassembler tools). https://github.com/orgs/gm-archive

Document History

Version	Date	Changes
v1.0	2026	Addition of RIGEL-WEB and updated serial protocol

End of RIGEL Windows PC Communication Protocol Technical Reference