`

A data-logging system

Robert R. Fenichel

Table of Contents

1 overview

I here describe a datalogging system that I developed for myself but that may be of interest to others.

The anticipated data will be of many different kinds, but they will be read at intervals of seconds or minutes, not milliseconds, with no datum requiring more than a few bytes to record.

The core of the system is a Windows Desktop Program, run intermittently, and a Teensy-based Base Station, connected to the desktop machine by USB and run continuously. The outer surface of the system consists of several **peripheral stations** (now eight**)** and **annunciators** (now two).

1.1 some choices made

1.1.1 desktop infrastructure

My desktop computer runs Windows 10.

The Desktop Program was developed with Delphi 7, which dates to 2002. More recent versions of Delphi eliminate a few bugs, but Embarcadero is trying to bring its Delphi and C++ compilers closer together, mostly by downgrading Delphi.

1.1.2 XBees

I am using my XBee transceivers in hub-and-spoke fashion, although the newest XBees allow implementation of mesh networks. A mesh network would in principle be more reliable. Switching over would require replacement of all my old XBP24s, and I haven't felt any need to do this.

All of the system's XBee traffic is carried out at 38 400 baud. Once or twice a day, data are garbled (and detected) when (I think) two devices try to transmit at once. Collisions might be even less frequent if a higher transmission speed were used, but RF noise might then start to cause trouble. I have not bothered to do pertinent experiments.

1.1.3 sensor support

In an earlier version of the system, each peripheral board carried signal-conditioning circuitry (for example, diodes and voltage dividers to allow untamed sensors to be connected to Teensy pins that did not tolerate voltages outside the [0, 3.3V] range). Each of these older peripheral boards also carried a small relay, allowing control of external effectors. In the newer boards, these task-specific components have been exiled to satellite boards, to be deployed and attached as needed.

The early sensor drivers were generalized. For example, a "resistance" driver handled photocells, the pond-depth sensor, and (in principle) any other sensor that provided a resistance as its value. Each of these old drivers had to individualize itself at run time, with messy code. The drivers are now more specific; there is a depth-sensor driver, a photocell driver, and so on.

1.1.4 Teensy pins

The system was first built with Teensy 3.5 development boards, and later moved to Teensy 4.1s. For ease of software maintenance and hardware replacement, the Teensy boards are socketed onto my project's PCBs, now with ZIF sockets.

The Teensy boards are *almost* breadboard-compliant: Most of their pins are in two parallel rows, 0.6″ apart with 0.1″ pin/pin spacing. A few pins are not placed in either of these rows, and I have wavered in my approach to these off-row pins.

In the earlier version of the system, I constructed *ad hoc* sockets to capture all of the Teensy pins that I thought I might ever make use of. These nonstandard sockets made the Teensys difficult to remove and replace without damage.

I briefly toyed with the idea of soldering the Teensys to breakout boards, thereby bringing all the pins of potential interest to two clean parallel rows of breadboard-ready pins.

Finally, I recognized that of all the off-row Teensy pins, only one (the **VUSB** pin) was likely to be useful to me. The on-row pins could be plugged into a breadboard or into a conventional socket on my PCB, while a short piece of hookup wire could be soldered to the **VUSB** hole on the Teensy, with its other end free to be connected to an off-row breadboard hole or plugged into a screw terminal on my PCB.

1.2 peripheral stations[1]

Each of the peripheral stations uses the same printed-circuit board design and the same software. Each peripheral-station PCB carries

- a Teensy 4.1 microprocessor,[2] with an intrinsic connector for a μSD memory card;
- connectors and regulators allowing the board to be powered from an unregulated 9-12 VDC source or from the Teensy's USB connector;
- a battery-backed DS1307 calendar clock;[3]
- an XBee radio transceiver;[4]
- a 3.2-inch, 320 x 480-pixel TFT display;[5]
- connectors for a variety of sensors and relays.

A peripheral station can be configured (by a file on its μSD card) to operate as a stand-alone datalogger, collecting data from its sensors, using the DS1307 to timestamp the data, and then storing the timestamped data on the μSD card. None of the peripheral stations is now being used this way.

As now configured, each of the peripheral stations uses its μSD card only for configuration settings and images received from attached cameras. Instead of timestamping and saving their sensor data, the peripheral stations use their XBee transceivers to send the data to the central **Base Station**.

1.3 annunciators[6]

Each **annunciator** is an output-only station, containing a Teensy 4.1, an XBee transceiver, and a 3.2-inch, 320 x 480-pixel TFT display, in and on an aluminum case.[7] The Teensy's μSD card is used to hold a collection of family photos. In operation, every few minutes the annunciator receives lines of text from the Windows application, forwarded by the base station. The annunciator goes through a cycle of showing a few photos, then showing the most recent received text, if it is not older than 2 hours. Every so often, it shows a special image (*e.g.*, a valentine-style heart, or a Canadian flag).

The board and program of the annunciators are straightforward. One annunciator is shown in Figure 1.

---

[1] See Section 5 for details.

[2] See https://www.pjrc.com/store/teensy41.html (accessed 2024-03-12).

[3] See https://datasheets.maximintegrated.com/en/ds/DS1307.pdf (accessed 2021-05-25).

[4] See https://www.digi.com/xbee (accessed 2021-05-25).

[5] See https://www.adafruit.com/product/2050 (accessed 2026-07-03). I rely on the **Adafruit_ILI9341** library to drive this display; to accommodate its full pixel dimensions, I had to modify **Adafruit_ILI9341.h**, changing the lines

```
#define ILI9341_TFTWIDTH 240  ///< ILI9341 max TFT width
#define ILI9341_TFTHEIGHT 320 ///< ILI9341 max TFT height
```

to

```
#define ILI9341_TFTWIDTH 320  ///< ILI9341 max TFT width
#define ILI9341_TFTHEIGHT 480 ///< ILI9341 max TFT height
```

[6] See Section 6 for details.

[7] See https://leeselectronic.com/en/product/10500-enclosure-instrument-case-le-453.html (accessed 2026-07-03).

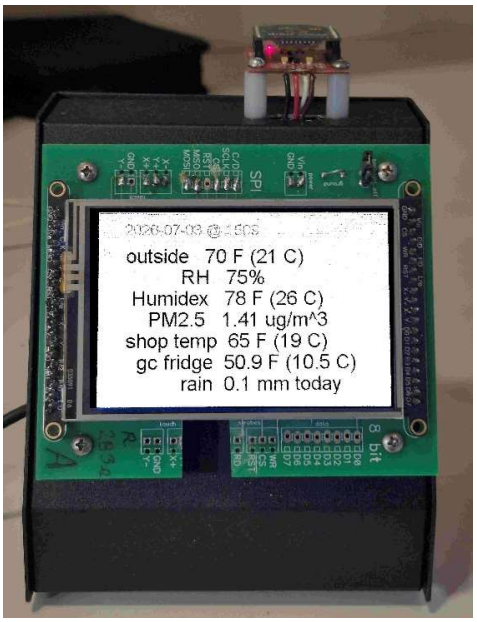


Figure 1 Annunciator

1.4 Base Station[8]

The Base Station, built on another home-designed printed-circuit board, carries

- a Teensy 3.5 microprocessor, with an intrinsic connector for a µSD memory card;
- connectors and regulators allowing the board to be powered from an unregulated 9-12 VDC source or from the Teensy's USB connector;
- a TeensyView[9] display;
- a battery-backed DS1307 calendar clock; and
- an XBee radio transceiver.

The Base Station is meant to run continuously, with occasional connections to the Desktop Program.

When the Base Station is not connected to the desktop computer, data received from the peripheral stations are timestamped and stored on the µSD card. When the Base Station is connected to the desktop computer, data stored on the µSD card are dumped to the desktop computer, after which data newly arriving at the Base Station are immediately passed through.

1.5 Desktop Program[10]

The **Desktop Program** saves the accumulating data in a relational database. Drawing on the database, the Desktop Program can display snapshots of recent sensor readings, graphs showing the sensor readings as functions of time, and miscellaneous statistics.

[8] See Section 7 for details.
[9] See https://www.sparkfun.com/sparkfun-teensyview.html (accessed 2026-07-07).
[10] See Section 8 for details.

2 sensors

2.1 AC voltage

To detect the presence of AC voltage, an adapter board[11] built around an MID400[12] provides an output DC voltage that is high when AC voltage is absent, low when AC voltage is present. The value of one of the resistors on the adapter board is specific to the anticipated AC voltage level.

2.2 airborne particulates

SPS30 particulate sensors[13] can be connected to measure the number-density (#/cm^3) of $PM_{0.5}$, $PM_{1.0}$, $PM_{2.5}$, $PM_{4.0}$, and $PM_{10.0}$ particles and the mass-density (μg/m^3) of $PM_{1.0}$, $PM_{2.5}$, $PM_{4.0}$, and $PM_{10.0}$ particles in ambient air. The SPS30 relies on standard UART communication.

2.3 alternating current

To measure the current being drawn by the two primary electrical circuits of my house, an adapter board[14] connects to two clamp-on current transformers and provides two outputs of DC voltage proportional to detected current.

2.4 air pressure

A BMP085[15] sensor is used to measure the ambient air pressure. The BMP085 also measures temperature. It relies on a standard I²C interface.

2.5 anemometer

My anemometer[16] allows wind speed to be measured by closing a switch for about one third of each revolution of a wind-driven wheel. Mechanical switch bouncing doesn't seem to occur, but the cabling to the anemometer picks up occasional microseconds-long disturbances that would sometimes trigger false counting by the peripheral board's Teensy. Debouncing[17] with an oscillator frequency of 1 kHz or so eliminates these false counts.

---

[11] See http://www.fenichel.net/pages/Indoor_Activities/electronics/datalogger/peripheral%20station/satellite%20boards/DipTrace%20Schematic%20-%201962%20AC%20detect.pdf (accessed 2024-03-15).

[12] See https://www.onsemi.com/pdf/datasheet/mid400-d.pdf (accessed 2021-05-25).

[13] See https://www.sensirion.com/en/environmental-sensors/particulate-matter-sensors-pm25/ (accessed 2021-05-25).

[14] See https://www.fenichel.net/pages/Indoor_Activities/electronics/datalogger/peripheral%20station/satellite%20boards/ACDC%202 (accessed 2024-03-15).

[15] See https://www.sparkfun.com/datasheets/Components/General/BMP085_Flyer_Rev.0.2_March2008.pdf (accessed 2021-05-25). These sensors are obsolete, but I have used them because I had a few in stock. I have no experience with the BMP180, but it is said to be pin-for-pin compatible.

[16] See https://www.argentdata.com/files/80422_datasheet.pdf (accessed 2021-05-25).

[17] See Section 4 below.

2.6 cameras

VC0706 motion-detector/cameras[18] capture activity around the outside of the house. The VC0706 relies on standard UART communication.

The images produced by a VC0706 are bulky, so even when a peripheral station is transmitting other acquired data to a base station, it will save VC0706 images locally on its μSD card.

2.7 CO_2 level

An SCD30[19] sensor can measure temperature, relative humidity, and CO_2 levels. Each of these sensors relies on a standard I^2C interface.

2.8 DC voltage

DC voltage is measured using the Teensy's built-in ADC. This means that the measured voltage must be limited before arrival to be in the [0, +3.3V] range.

2.9 depth

To measure the depth of a small pond in my back yard, I use a sensor from Milone Technologies.[20] The Milone sensor is a clever pressure-to-resistance transducer with a built-in voltage divider, so scaling is all that's left for the sensor driver to do.

2.10 light

The photocells[21] I use vary in their resistance from a few hundred ohms (light) up to megohms (dark).

2.11 rainfall

Every time its bucket tips, my rain gauge[22] closes a switch for about 100 ms. Mechanical switch bouncing is minimal, but the cabling to the gauge picks up occasional microseconds-long disturbances that would sometimes trigger false counting by the peripheral board's Teensy. Debouncing[23] with an oscillator frequency of 100 Hz or so eliminates these false counts.

2.12 temperature, relative humidity, and dewpoint

I use DHT22[24] and SHT45[25] temperature-humidity sensors to measure temperature and relative humidity; my software computes the dew point and Humidex whenever a temperature and a relative humidity are available.

---

[18] See https://www.itead.cc/wiki/VC0706_UART_Camera_%EF%BC%88Supports_JPEG%EF%BC%89 (accessed 2021-05-25).

[19] See https://www.sensirion.com/fileadmin/user_upload/customers/sensirion/Dokumente/9.5_CO2/Sensirion_CO2_Sensors_SCD30_Datasheet.pdf and https://www.sparkfun.com/products/15112 (both accessed 2021-09-26).

[20] See https://milonetech.com/p/about-etape (accessed 2024-03-09).

[21] https://media.digikey.com/pdf/Data%20Sheets/Photonic%20Detetectors%20Inc%20PDFs/PDV-P9203.pdf (accessed 2021-05-25).

[22] See https://www.argentdata.com/catalog/product_info.php?products_id=168 (accessed 2021-05-25).

[23] See Section 4 below.

[24] See https://www.adafruit.com/product/385 (accessed 2021-05-25).

[25] See https://www.adafruit.com/product/5665 (accessed 2026-07-03).

The DHT22 requires only 3 wires for its interface, but it comes in a package with 4 spaced leads. On each peripheral board, a 4-conductor connector allows a DHT22 to be connected directly to the station with no cabling.

One of my DHT22s, potted in about 100 ml of silicone sealant,[26] is used to measure the temperature of the back-yard pond. It functions well in this environment; its response time is sluggish, but the water temperature of the pond changes only slowly anyway.

After a year or so of use, it is not rare for a DHT22 to lose its humidity sensor. When this happens, at least in my experience, the temperature sensor is unaffected.

The SHT45 has a standard I²C interface, and its humidity sensor, like that of the DHT22, can degrade into uselessness after prolonged exposure to high humidity. The superpower of the SHT45 is an intrinsic controllable heater that can be used to dry out the humidity sensor.

2.13 thermostat intent

My household thermostat can be manually set to Heat, Cool, or Off. A simple homebrew board reads this state using photocells apposed to LEDs on the thermostat's control box. The board's outputs are a DC level that is high when the thermostat's mode is Heat, and another DC level that is high when the mode is Cool.

2.14 wind direction

My wind vane[27] presents its reading as one of 16 resistances between 891Ω and 120K. The resistances are scrambled, so adjacent directions do not have adjacent resistances.

3 relays

Each older peripheral board included a small relay. These relays could switch only low currents, so in practice they were used to control the coils of off-board larger relays.

The relays now used are hefty SPDT relays[28] that I happened to have around, each mounted on its own little PCB. These relays require a 12V coil current, so the relay PCB[29] needs 12V power.[30] The board's circuitry allows the relay to be controlled by any of the Teensy's 3.3V current-limited output pins. Also, a three-position switch on the relay PCB allows the relay to be manually forced on, forced off, or left under control of the attached peripheral station.

The relay board can be configured so that the switched circuit connects

- the common ground and the switched 5V supply,
- the common ground and the switched 12V supply, or

---

[26] See https://www.homedepot.ca/product/mono-silicone-pro-clear-premium-silicone-rubber-kitchen-bath-plumbing-sealant-290ml/1001001157 (accessed 2021-05-25).

[27] See https://www.argentdata.com/files/80422_datasheet.pdf (accessed 2021-05-25).

[28] See https://media.digikey.com/pdf/Data%20Sheets/Tyco%20Electronics%20P%20B%20PDFs/Potter&Brumfield%20RELAYS.pdf (accessed 2024-03-09) for type **`RKA-5DG-12`**.

[29] See https://www.fenichel.net/pages/Indoor_Activities/electronics/datalogger/peripheral%20station/satellite%20boards/DipTrace%20Schematic%20-%20Potter-Brumfield%20relay%201964.pdf (accessed 2024-03-15).

[30] The peripheral boards can run from the power provided by a USB connection, but most of the time their **`Vin`** (`+5V`) rails are derived from a regulator driven by a 12V source, so 12V power is available here and there on the peripheral boards, labeled "wall power."

- an independent circuit (often, mains voltage) not connected to the common ground.

Relays are used to control a recirculating waterfall in the backyard pond, a nightlight in the living room, and a heater in the shop.

4 signal conditioning

(In the earlier version of the system, circuitry on the peripheral PCBs provided various level-shifting, diode protection, and debouncing. These functions, used only here & there, have been deported to satellite boards, deployed as needed.)

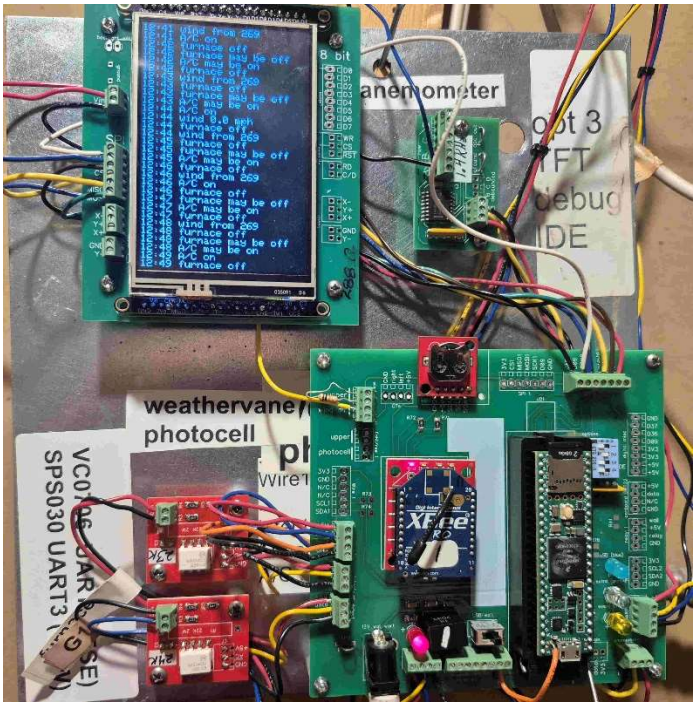
The Teensy's change-triggered interrupts can sometimes be triggered by ephemeral noise. Signals that might carry such noise are conditioned by debouncer boards[31] on their way to the Teensy.

Each debouncer board is built around a MC14490 chip.[32] The MC14490 accepts a digital input signal and maintains an internal oscillator whose frequency is set by an external capacitor. Its output is identical to the input signal, except that input LOW levels are passed through only if they are longer than 4 times the oscillator period.

For example, my rain gauge closes a circuit for about 100 ms at a time, so the oscillator on the board used there is set to about 100 Hz, rejecting pulses shorter than 40 ms or so. My wind gauge can generate a square wave of up to 50 Hz or so, so the oscillator used there is set to about 1 kHz.

5 peripheral stations

Here is a picture of a sensor-heavy peripheral station.



[31] See https://www.fenichel.net/pages/Indoor_Activities/electronics/datalogger/peripheral%20station/satellite%20boards/DipTrace%20Schematic%20-%201959%20quad%20debouncer.pdf (accessed 2024-03-15).

[32] See https://www.onsemi.com/pdf/datasheet/mc14490-d.pdf (accessed 2024-03-09).

Figure 2, peripheral board in situ

5.1 hardware

5.1.1 PCB

Each peripheral-station printed-circuit board looks like this:

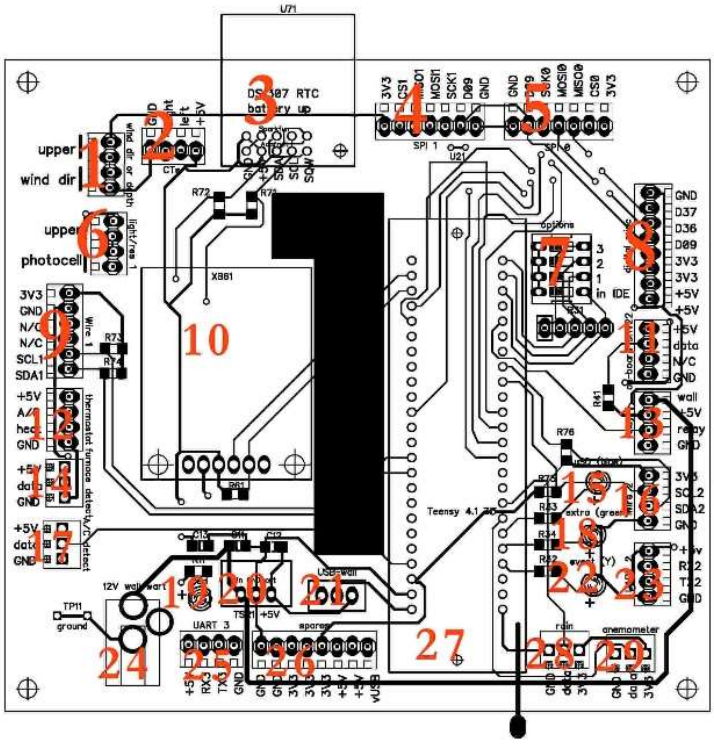


Figure 3 peripheral-station PCB

The DipTrace schematic and layout files are on available on my Web site.[33]

5.1.2 Core circuitry

The Teensy CPU is here at **27**. If the Teensy's as-provided **`Vin/VUSB`** connection has been severed, a wire from the Teensy's off-row **`VUSB`** pin can be plugged into the rightmost pin at **26** ("spares") to provide power.

The µSD card of the Teensy is formatted as a FAT32 volume.

The connectors at **4** and **5** allow access to the Teensy's SPI ports; at each peripheral station, the display is connected to **`SPI 0`** at **5**.

[33] See https://www.fenichel.net/pages/Indoor_Activities/electronics/datalogger/peripheral%20station/ (accessed 2026-07-12).

The DS1307 clock is attached at **3**, connected to the Teensy's **`Wire`** channel. The available DS1307 boards from Adafruit[34] and Sparkfun[35] have different pinouts, so provision is made for using either one here. Using controls on its Dashboard,[36] the Desktop Program can set all of the peripheral DS1307s. These clocks are maintained on local time, so they need to be set forward and backward as Daylight Savings Time comes and goes.

The odd central blank shape, black in this image, is white in the actual PCBs. The shape is handy for board-specific handwritten annotations.

An XBee breakout board is attached at **10**, connected to the Teensy's **`UART8`** channel. The XBees in my system are old XBP24s and a few newer XBee3s.[37] I use Digi's XCTU utility[38] to configure the XBees. The peripheral-station XBees and the Base-Station XBee all have the same net identifier (**`ID`**, in XBee lingo), and each of the XBees has a unique unit identifier (**`MY`**). The XBee node identifiers (**`NI`**) are used for inventory purposes, but the datalogger software does not rely on them.

When the lowermost switch at **7** is on, the peripheral station believes that it is running in the Teensyduino IDE. This enables a variety of debugging messages to be sent to the IDE for display. The other switches here can be used in debugging.

The yellow LED at **22** flashes whenever one of the attached sensors has provided data. The blue LED at **15** is illuminated when there are data waiting in the µSD memory. The green LED at **18** is used for debugging. The red LED at **19** is illuminated whenever +3.3V power is available.

5.1.3 power-related

The switch at **21** selects the board's power source, either the barrel connector ("wall power," **24**), or the Teensy's USB connector. When wall power is used, the board's +5V rail is fed by the voltage regulator at **20**.

The connector at **26** provides access to ground, +3.3V, and +5Vconnectors for off-board components

5.1.3.1 UARTs

The connectors at **23** and **25** provide access to the Teensy's **`UART3`** and **`UART2`** channels, respectively. They could be used for any UART-dependent devices, but I now reserve them for an SPS30 particulate sensor[39] and a VC0706 camera.[40]

5.1.3.2 digital input

The connectors at **12** ("thermostat"), **14** ("furnace detect"), and **17** ("A/C detect") allow digital input. They could be used for pushbuttons, latched switches, or any other generators of binary data, but I now reserve them as follows:

---

[34] See https://www.adafruit.com/product/3296 (accessed 2026-07-03).
[35] See https://www.sparkfun.com/sparkfun-real-time-clock-module.html (accessed 2026-07-03).
[36] See Section 8.2.1 below.
[37] The XBee form factor has not changed, so breakout boards obtained years ago can be used with modern XBee units. The old XBP24 models are no longer available.
[38] See https://www.digi.com/products/embedded-systems/digi-xbee/digi-xbee-tools/xctu (accessed 2021-05-25).
[39] See Section 2.6 above.
[40] See Section 2.5 above.

- The connector at **12** gets levels that tell whether my home thermostat is in cooling mode, heating mode, or neither.
- The connectors at **14** and **17** get levels that show activity of the house's furnace and air-conditioner, respectively.

The connector at **8** ("digital misc") connects to various Teensy pins and power rails. It is not used in any of the existing boards.

The connector at **11** is configured to allow direct local connection of a DHT22.

The connectors at **28** and **29** are reserved for my rain gauge[41] and my anemometer,[42] respectively. As noted with the descriptions of those sensors, these connections pass through debouncers before getting to the peripheral PCB.

5.1.3.3 digital output

The connector at **13** ("relay 1") provides access to digital output from the Teensy. The "wall" and +5V power pins on this connector facilitate its use with the relays described in Section 3 above.

5.1.3.4 I²C devices

The connectors at **9** ("Wire1") and **16** ("Wire 2/BMP085") provide access to the Teensy's **`Wire1`** and **`Wire2`** I²C channels, respectively. These connectors are used with CO2 sensors,[43] SHT45s,[44] and BMP085s.[45] Some I²C devices have their signals supported with LTC4311 terminators.[46]

5.1.3.5 voltage input

The connector at **2** ("CTs") is arranged to accept two voltage inputs. It is used with current transformers to allow for monitoring of the power used by the house.

5.1.3.6 resistance input

The connectors at **1** ("wind dir or depth") and **6** ("light/res") allow the Teensy to measure resistance. Each is arranged so that the input resistance is connected as the lower arm of a voltage divider, with the upper arm ("upper") provided by a sensor-specific resistor.

5.2 software

The software of the peripheral stations is about 7000 lines of C++.

5.2.1 the **`X_config.txt`** file

Every peripheral station, freestanding as a datalogger or connecting to a base station, sensor-heavy or sensor-light, runs the same program. Different peripheral stations behave differently because of different configuration information, made available in a text file on the µSD card called **`X_config.txt`**.

---

[41] See Section 2.11 above.
[42] See Section 2.5 above.
[43] See Section 2.7 above.
[44] See Section 2.12 above.
[45] See Section 2.4 above.
[46] See https://www.adafruit.com/product/4756#technical-details (accessed 2026-07-04).

The **X_config.txt** file is an unordered sequence of lines, although lines are conventionally grouped by function. Blank lines are ignored, as are lines (or portions of lines) preceded by **//** (two forward slashes).

Each non-comment line of the file is of the form

<category>.<name> = <value>

where any number of blanks may precede and/or follow the equals sign. For example, the lines

```
Analog.NBits          =  12   // 8, 10, 12, or 13
Board.LoggingToUSD    =   0
Board.SendingToBase   =   1
Board.msBlinkLength   = 200
```

tell the board

- to use 12 bits in the Teensy ADCs;
- that received data should be sent to the base station and not saved to the μSD card; and
- that when data are received from a sensor, the signaling LED at **20** should stay on for 200 ms.

The values thus assigned can be integers, fixed-point numbers, floating-point numbers, or quoted strings. Boolean values are indicated with 0 (false) and 1 (true). Some of the values needed are times of day, expressed as integers, *100* × <*hour*> + *minutes*. Other values needed are time durations, expressed in milliseconds or microseconds, so suffixes are recognized to make large numbers less unwieldy. For example, **X_config.txt** in one of my peripheral stations includes the lines

```
// ********************** TV room waterfall
Waterfall.InUse                       =  1
Waterfall.LightHigh                   = 35
Waterfall.LightLow                    = 32
Waterfall.msMinimumHold               =  1m
Waterfall.msStandardReportInterval    =  5m
Waterfall.SensorTag                   =  'r'
Waterfall.Singleton                   = 0
Waterfall.TempHigh                    = 41
Waterfall.TempLow                     = 39
Waterfall.usStandardReadInterval      = 15KK


//  ************************* (weathervane)
WeatherVane.InUse                     =  0
WeatherVane.Damping                   =  0.9
```

Here, the **s** suffix and **K** suffix are each factors of 1000, so that the relay's status is read every 15 000 000 microseconds, or (more intelligibly) every 15 seconds. The **m** suffix = 60**s**; an **h** suffix (not shown) = 60**m**.

The peripheral-station program creates a driver module for every possible sensor, so an important function of **X_config.txt** is to specify which drivers should actually be run. In the example just quoted,

the peripheral station is told that it must control a relay (**`Waterfall.InUse = 1`**), but it is not monitoring a weathervane (**`WeatherVane.InUse = 0`**).

5.3 XBee support

The main job of the peripheral station's XBee transceiver is transmission of data to the Base Station. For this purpose, a line like

```
XBee.BaseID = 200  // (0xC8)
```

is used to tell the peripheral station the unit identifier (**`MY`**) of the Base Station.[47]

The peripheral station also listens for XBee input from the Desktop Program, passed on by the Base Station. For example, the station listens for a message giving it a new setting for its DS1307 clock. The SCD30 CO_2 sensors need to know the current ambient pressure, so another occasional message broadcasts the pressure reading from the system's one BMP085. If the Desktop Program has sent a message intended for an inactive component (for example, if the message is directed to the relay of a peripheral station that has no relay), then the message is simply discarded by the XBee support code.[48]

5.4 data handling

If a peripheral station were always to be used as a freestanding datalogger, and if it carried only a single sensor (say, a thermometer), then it could use its µSD card to record data in a dense binary encoding, keeping only a few bits of change data for each timestamp and each noted temperature.

The situation is different when a given station can carry multiple sensors, of the same or different types, with a given sensor possibly supplying a vector of multiple values each time it is read, and when multiple peripheral stations, not coordinating with each other, are all transmitting data to the Base Station for timestamping and forwarding (possibly after storage at the Base Station) to the Desktop Program.

Coordination is achieved by arranging that certain files are included at compile time by the peripheral-station program, the Base-Station program, and the Desktop Program. These files are written so as to be usable in C++ (for the peripheral stations and Base Station) and in Delphi (for the Desktop Program). These files become definitions of enumerations in C++ and matching enumerated types in Delphi.

5.4.1 board slots

The text of the **`BoardSlots.Inc`** file is

```
//  0         1              2            3      4           5
bsBogus, bsACDetect_1, bsACDetect_2, bsBMP, bsDepth, bsDHT22_1,
bsDHT22_2, bsLightning, bsRelay_1, bsResistance_1,
bsResistance_2, bsResistance_3, bsSPS30, bsSwitch_1, bsSwitch_2,
bsVC0706, bsVoltage_1, bsVoltage_2, bsRelay_2, bsRelay_3,
bsSwitch_3, bsSwitch_4, bsWire_1, bsVoltage_3, bsWire_2,
bsAnemometer, bsRainGauge, bsACIntent, bsHeatIntent,
bsBoardReboot
```

It has one entry for each of the potential sensor (or relay) connections to the peripheral board. The codes here greyed out are not used, but they are preserved for backwards compatibility. If different or expanded peripheral boards were added to the system, additional entries could be made in this file.

---

[47] XBees communicate in hexadecimal, but — as shown here — XBee addressing within the datalogging system is expressed in decimal.

[48] For more on the Desktop-to-Base messages, see Section 7.2.1 below.

5.4.2 WhatMeasured

The text of the **WhatMeasured2.inc** file is

```
//   0          1                2              3       4
wm2Bogus, wm2Undefined, wm2PressureInches, wm2RH, wm2TempF,
wm2DepthInches, wm2LightningDistance, wm2LightningDistPerHour,
wm2LightningEnergy, wm2LightningFlashes,
wm2LightningNoisePerHour, wm2IsNowOn, wm2Resistance, wm2MassPM1,
wm2MassPM25, wm2MassPM4, wm2MassPM10, wm2NumPM05, wm2NumPM1,
wm2NumPM25, wm2NumPM4, wm2NumPM10, wm2PartSize, wm2EventCount,
wm2Frequency, wm2FrequencyMax, wm2FrequencyMin, wm2FrequencyRaw,
wm2Jammed, wm2NewEvents, wm2SwitchClosed, wm2TargetF,
wm2VC0706Motion, wm2VC0706Snap, wm2Voltage, wm2DewpointF,
wm2DeviceFailure, wm2FailureDatum, wm2RelayAck, wm2VC0706Manual,
wm2HumidexF, wm2CO2ppm, wm2RainMMPerHour, wm2Calibrate,
wm2SpeedupChanged, wm2Prototype, wm2Debugging, wm2OnOffIntent
```

These entries are used to tag recorded data. They are mostly associated with specific sensor types, but some codes are used by the system for administrative purposes not directly related to sensor data. The codes here greyed out are not used, but they are preserved for backwards compatibility.

5.4.3 triples

The most frequent use of the **WhatMeasured2** elements is in encoding individual sensor data. Every non-discarded sensor datum passes through a stage of being recorded as a **triple**, of the form

<wm2 index>TAB<mantissa>TAB<negexp>

where TAB is the ASCII tab character (**\t** in C++) the <mantissa> and <negexp> are (possibly signed) integers, and the datum represented is <mantissa> × $10^{-<negexp>}$. For example, a temperature reading[49] of 65.42 °F might appear as the triple

```
4\t6542\t2
```

5.4.4 sensor tags

For local purposes of the peripheral station, each sensor is assigned a single-character tag, mnemonic if possible. Different stations can use different sets of tags.

5.4.5 board tag

Each peripheral station or annunciator has a unique, single-letter board tag, assigned in **X_config.txt** with a line like

```
Board.Tag  = 'S'
```

5.4.6 formatting for storage on the peripheral-station µSD card

The data files on a peripheral station's µSD card are text files, each named <YYYYMMDD>**.txt** to show when it was created. If **FileSystem.NewFileNameQDay** is set to 1 in **X_config.txt**, then a new data file is started on each calendar day. Each line of the file is of the form

[49] Temperature values throughout the system are handled in Fahrenheit, only because it is then more convenient to graph temperature and relative humidity on the same scale.

<YYYY-MM-DD HH:MM:SS>**\t**<board slot>**\t**<triple>

so that the temperature reading described in Section 5.4.3 above might (if it had been made by the DHT22 connected at **bsDHT22_1**) appear as

2021-05-07 13:18:43\t5\t4\t6542\t2

5.4.7 formatting for transmission to the Base Station

A sensor may provide more than one datum with each reading. Also, a sensor driver might accumulate data for a while before reporting a set of statistics describing these data. A peripheral station sends data to the Base Station only after a sensor driver has indicated that one or more triples are ready to be packaged into a **report**. Each report is a single ASCII line of the form

<board tag>**\t**<board slot>**\t**<triple 1>**\t**<triple 2>**\t . . .\t**<triple N>**\t**<checksum>

so the data from a single BMP085 reading (slot 3, 29.46 in Hg, 64.6 °F) from the board with tag **S** might be transmitted as

S\t3\t2\t2946\t2\t4\t646\t1\t3604

5.4.8 reading & reporting

Some sensors provide noisy data that cannot be usefully interpreted until multiple readings are averaged. In other cases, the most informative reporting will describe how sensor readings have varied over the recent past. In the **X_config.txt** file, **usNominalReadInterval** and **msNominalReportInterval** lines can specify that a given sensor is read at certain intervals, but reported only at different (longer) intervals.

For example, the **X_config.txt** lines

```
photocell.Damping                  =  0.9
photocell.usNominalReadInterval    =  1mK
photocell.msNominalReportInterval = 10m
```

might be used to specify that a photocell should be read every minute or so, but the data (exponentially damped as specified) should be reported only every 10 minutes or so.

The "nominal" and "or so" in the previous paragraph refer to the fact that the actual reading and reporting intervals are slightly randomized around the specified intervals. The randomization is meant to reduce the incidence of collisions among separate peripheral stations' XBee transmissions.

A sensor driver may impose sensor-specific limits (typically, a minimum read interval) to override a specified read interval. Also, the Desktop Program can send a peripheral board a message directing it to speed up its reading and reporting by a factor of 10, but still subject to the minimal read intervals just mentioned.

5.4.9 the **tSensor**[50] class

As part of its initialization, the peripheral-station program loads a driver for every possible sensor. Guided by **InUse** lines in the **X_config.txt** file, selected drivers are marked as being active. The main activity of the central loop of the peripheral-station program is to call the **loop** method of each active driver.

[50] My C++ code borrows the Delphi convention of having class names ("object" names in Delphi) begin with **t**.

Each sensor driver is an instance of a class specific to one type of sensor. Thus, there is a **tBMP085** class, a **tRainGauge** class, and so on. More could be added. Each of these classes is a descendant of the **tSensor** class.

Most of the methods of the **tSensor** class are protected, accessible to only **tSensor**'s descendants. **tSensor**'s only public methods are **loop, ShutSensorDown,** and the virtual methods **setup** and **ReadTheSensor**.

The **setup** method, implemented by each instance of each driver routine, associates the instance with a name and (usually) a Teensy pin and a board slot. This method is also responsible for organizing hardware and software initialization, guided by the peripheral station's **X_config.txt** file. Much of most **setup** methods consists of note-taking that may later be useful for debugging.

Every active sensor's **loop** method is called on every circuit of the peripheral station's central **loop** method. If the sensor is due to be read (per **usNominalReadInterval**) , then the sensor's **loop** calls the sensor's **ReadTheSensor** method.

5.5 sensor drivers

5.5.1 anemometer

The core of this driver is an interrupt-service routine linked to a Teensy pin. The routine counts periods in which the pin has been pulled low for longer than the number of microseconds specified by **usMinLowTime** in the peripheral station's **X_config.txt** file. Each such period corresponds to one revolution of the anemometer. Guided by the **X_config.txt** file's **WindMPHPerHz, Damping,** and **MaxPlausibleHz** parameters, the sensor uses the **wm2Frequency, wm2FrequencyMin, wm2FrequencyMax,** and **wm2FrequencyRaw** tags to report the recent damped wind speed (MPH), minimum recent speed, max recent speed, and undamped speed, respectively.

5.5.2 BMP085

Most of the BMP085 driver is code copied from a Teensy forum.[51] Reports from this driver use the **wm2TempF** and **wm2PressureInches** tags.

5.5.3 depth

This driver reads the voltage coming from the Milone Technologies sensor. Using the **Damping, DryAnalogReading, DryInches, WetAnalogReading,** and **WetInches** parameters from the **X_config.txt** file, the sensor uses the **wm2DepthInches** tag to report the damped estimate of depth.

5.5.4 DHT22

Most of the work of the DHT22 driver is done by the Teensy's **DHT** library routine. The values read from the sensor are accumulated between reports with exponential damping. Reports from this driver use the **wm2TempF** and **wm2RH** tags.

5.5.5 heater control

This driver is connected at **13** to a satellite board[52] whose relay controls a heater. A DHT22 on the same peripheral board follows the local temperature. The desired behavior of the heater is to keep the local temperature above **TempLow** all the time, and to keep it above **TempHigh** between the hours of

[51] See https://forum.pjrc.com/threads/17143-I2C-barometer-BMP085?highlight=bmp085 (accessed 2021-05-25).
[52] See Section 3 above.

MilTimeOn and **MilTimeOff**. To avoid relay-rattling, the driver enforces a minimum time-in-state (**msMinimumHold**). Also, the target temperature (**TempLow** or **TempHigh**) is interpreted with a hysteresis rule: The driver will turn the heater off only when the temperature is **HysteresisF** above the target temperature, and it will turn the heater on only when the temperature is **HysteresisF** below the target temperature.

This driver uses the **wm2TargetF** tag to report the current target temperature. On the same peripheral board, a voltage detector uses a **wm2IsNowOn** tag to report the status of the heater.

5.5.6 nightlight

This driver controls a relay, so that an attached lamp can be on when and only when

- the local light level is low enough, and
- the local time is within a specified bracket.

The time bracket (**MilTimeOff, MilTimeOn**) is specified in the **X_config.txt** file, together with a parameter (**msMinimumHold**) that keeps the driver from rattling the relay. Within the specified time bracket, the relay is turned on only when the local light level is below **LightLow**, and it is turned off only when the local light level is above **LightHigh**. The driver uses the **wm2IsNowOn** tag to report the status of the relay.

5.5.7 photocell

This driver links a photocell, connected as the lower arm of the voltage divider at 6, to an analog pin of the Teensy. The upper arm of the divider is specified with the **UpperOhms** parameter in **X_config.txt.** The driver then scales the reading into the [0, 100] range, and any value less than **CalledBlack** is scaled to zero. A **Damping** parameter is also used. The damped, scaled value is reported with the **wm2Resistance** tag.

5.5.8 power

Each of the current transformers connected at 2 is associated with a separate instance of this driver. To estimate the current seen by the associated CT, the driver depends on empirical constants provided in the **X_config.txt** file. The estimated current (amperes) is computed as

MultiplyBy × <Input voltage> / **DividerRatio** + **ThenAdd**

This is reported using the **wm2Voltage** tag.

5.5.9 rain gauge

The core of this driver is an interrupt-service routine linked to a Teensy pin. The routine counts periods in which the pin has been pulled low for longer than the number of microseconds specified by **usMinLowTime** in the peripheral station's **X_config.txt** file. Each such period corresponds to one tip of the rain gauge's bucket. No more than **MaxNewEvents** can be counted for one report. This driver uses the **wm2EventCount** tag to report the count.

5.5.10 SCD30

This driver tries to provide its associated SCD30 with the current air pressure as broadcast from time to time from the Base Station, but uses a default value (30 in Hg) if no current value is available.

The SCD30 data are reported with the **wm2CO2ppm, wm2RH**, and **wm2TempF** tags.

5.5.11 SHT45

This driver protects its SHT45's humidity sensor by taking special action after it has reported a relative humidity greater than the **X_config.txt** file's **DangerousRH** parameter. When next asked to read the SHT45, the driver instead instructs the SHT45 to turn its heater on at 200 mW for one second. Subsequent read requests are ignored until **msEquilibration** milliseconds after the heating pulse; this parameter is empirically chosen to be sufficient for recovery of the SHT45's temperature sensor. This driver also uses the **Damping** parameter, and it reports using the **wm2TempF** and **wm2RH** tags.

5.5.12 SPS30

The SPS30 device has an internal fan that can be activated at intervals to clean the device's innards. The SPS30 driver depends on **X_config.txt** parameters (**AutoCleanDisabled, AutoCleanIntervalHours, AutoCleanImmediately**) to organize the scheduling of these sanitation events.

The SPS30 data are reported with the **wm2MassPM1, wm2MassPM25, wm2MassPM4, wm2MassPM10, wm2NumPM05, wm2NumPM1, wm2NumPM25, wm2NumPM4, wm2NumPM10,** and **wm2PartSize** tags.

5.5.13 VC0706

Most of the work of this driver is done by the driver provided in the AdaFruit library.

The VC0706 camera will record 160×120 images unless one of the **640x480** or **320x240** parameters is true. Either of the larger images takes about 19 seconds to store; the 160×120 image takes about 5 seconds. If **MotionDetect** is true, then the camera will take and store an image whenever motion is detected. Otherwise, the camera will take and store an image whenever a reporting interval has passed.

Image files stored on the μSD card are named **JPEG0000.JPG, JPEG0001.JPG**, and so on. Whenever an image is stored, the driver reports its file number, using one of the tags **wm2VC0706Motion** and **wm2VC0706Snap**.

As described in Section 8.6 below, it's also possible to direct the taking of a VC0706 image from the Desktop Program.

5.5.14 voltage detector

The MID 400 chips used for AC detection send DC analog output to the Teensy, higher when AC voltage is absent, lower when it is present. The analog levels are usually close to one rail or the other, so the value in an **X_config.txt** line like

```
ACDetect_1.ACPresentIfBelow  = 0.5 // volts
```

is not critical. The board that reads the mode of the household thermostat (Section 2.13 above) is similar, except that the "heat intent" and "cool intent" modes are indicated by high levels instead of low levels. Each instance of this driver reads a Boolean **TrueIfBELOWThreshhold** parameter to see how to interpret the received level. Reports from this driver use the **wm2IsNowOn** tag.

5.5.15 waterfall

This driver controls a relay, so that a pump in an outdoor pond can be on when and only when

- the outside light level is high enough, and
- the outside air is warm enough.

To prevent relay-rattling, the **X_config.txt** file provides the **msMinimumHold** parameter. Hysteresis is also reinforced by defining two thresholds for each of light and air temperature. The relay is turned on only when

- the outside light level is above **LightHigh**, and
- the outside air temperature is above **TempHigh**.

The relay is turned off when

- the outside light level is below **LightLow**, or
- the outside air temperature is below **TempLow**.

Otherwise, the relay's state is unchanged. The driver reports the relay's state using the **wm2IsNowOn** tag.

5.5.16 weathervane

As noted in Section 2.14 above, my weathervane presents a resistance to encode a direction. The weathervane is connected to the peripheral board using the voltage divider at **1**, and the upper arm of the divider is a resistor described in **X_config.txt** with the **UpperOhms** parameter. This driver turns the observed resistance into a value in [0, 360), where 0° is North, increasing clockwise. To compute this value, the driver makes use of specifications provided in the vane's data sheet,[53] which describes the (highly nonmonotonic) relation between the vane's direction and the resistance of the connection.

To perform exponential damping that combines several readings of wind direction before reporting, a **Damping** parameter is used, and the scalar directions are temporarily converted to $(W\cos\theta, W\sin\theta)$ vectors, where W is the most recent measured wind speed (or 1, if no recent wind speed is available). The damped vector is converted back to a directional scalar, which is reported using the **wm2Resistance** tag.

6 annunciators

The annunciator software (about 200 lines of C++) is elementary. It receives text from its XBee, with each line either

- specifying how many lines of text to anticipate, or
- providing a line of text, along with its intended size and color.

When the main loop of the program sees that

- it has a full set of lines to be displayed, and
- sufficient time (5 minutes) has passed since the last update

it clears the screen and displays the waiting lines.

[53] See https://www.argentdata.com/files/80422_datasheet.pdf (accessed 2021-05-25).

7 Base Station

7.1 hardware

The current Base Station PCB still uses a Teensy 3.5. It looks like this:

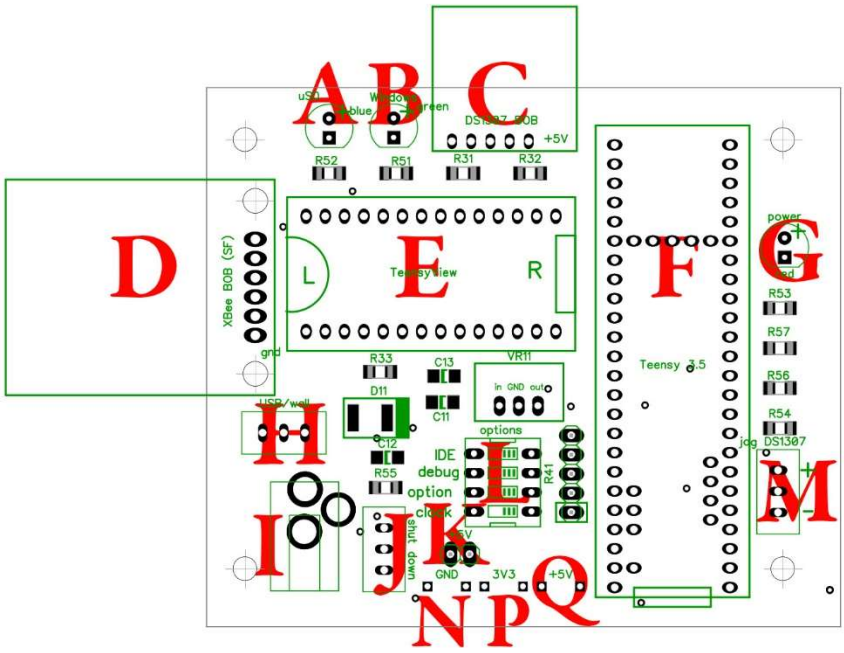


Figure 4 Base-Station PCB

The schematic and the DipTrace schematic and layout files are available on my Web site.[54]

Many of the components here (XBee at **D**, barrel connector at **I**, ground testpoint at **N**) are the same as those used in the peripheral stations. Testpoints for the +3.3V and +5V rails are provided at **P** and **Q**, respectively, and an ammeter shunt for the +5V supply is provided at **K**. The Teensy 3.5 is at **F**.

A TeensyView display unit is present at **E**.

The Base Station's DS1307 clock (here at **C**) is maintained on Universal Time; the hassle of Daylight Savings Time is handled in the Desktop Program.

The blue LED at **A** is lit when data from the peripheral stations are waiting on the Teensy's µSD card. The green LED at **B** is lit when the Base Station has an open connection (through the Teensy's USB port) to the Desktop Program. The red LED at **G** is lit whenever the PCB has power.

The two-position switch at **H** selects the board's power source, either the barrel connector at **I** or the Teensy's USB port.

If the switch at **J** is closed, then the Base Station will initiate an orderly shutdown.

[54] See https://www.fenichel.net/pages/Indoor_Activities/electronics/datalogger/base%20station/ (accessed 2021-05-26).

7.2 software

The central loop of the Base-Station software (about 2000 lines of C++) reads as many characters as it can from the USB port (that is, from the Desktop Program), and then from the XBee (that is, from the peripheral stations). Whenever a complete message is identified from either source, it is processed.

7.2.1 messages to and from the Desktop Program

Each message from the Desktop Program to the Base Station is a tab-delimited ASCII string; the tokens are decimal integers. Many of the tokens are indexes into the **WindowsToBase** enumeration; the text of the **WindowsToBase.inc** file is[55]

```
   wtbEmpty, wtbConnect, wtbDeleteJPEGs, wtbDisconnect,
wtbInitializeuSD, wtbRelayAction, wtbRelayHysteresis,
wtbRelayLight, wtbRelayMode, wtbRelayOff, wtbRelayOn,
wtbRelaySchedule, wtbRelayTemps, wtbRelayWhichTemp, wtbSendToXBee,
wtbSetTime, wtbShutDown, wtbStartDump, wtbTakeVC0706Image,
wtbTargetF, wtbThermostat, wtbThermostatRelax,
wtbTickleBaseStation, wtbAnnunciator, wtbBroadcastPressure,
wtbBroadcastOutsideTempF, wtbBroadcastLocalTime,
wtbSpeedupChanged, wtbEnableELWire, wtbReadTime, wtbDebugging
```

Messages from the Base Station to the Desktop Program are also tab-delimited ASCII strings. Some of these messages simply package up the data provided by the peripheral stations, while others are various status reports. The text of the **BaseToWindows.inc** file is[56]

```
   btwUnknown, btwDumpData, btwLiveData, btwDumpEnded,
btwDumpingBlock, btwSavingBuffer, btwAlreadyConnected,
btwConnected, btwDisconnectedBS, btwDisconnectedRQ,
btwDumping, btwFileSystemTrouble, btwInterrupted,
btwJustConnected, btwLoopTiming, btwPassThrough, btwSetupComplete,
btwStartingFileSystem, btwTimedOut, btwStartingXBee,
btwXBeeBufferBusy, btwXBeeCommandIssued,
btwXBeeDifficultCommandMode, btwXBeeInCommandMode,
btwXBeeMYFailed, btwXBeeNetIDFailed, btwXBeeNIFailed,
btwXBeeNoCommandMode, btwXBeeNoSetDestination, btwXBeeOK,
btwXBeeSerialNumberFailed, btwXBeeSettingDestination,
btwXBeeStuckInCommandMode, btwFailedTransmission, btwNotDeadYet,
btwNotDuringDumping, btwProcessingRequest, btwReadTime,
btwDebugging
```

7.2.1.1 connection-related

The Base Station and the Desktop Program are both continually looking for reassurance that they are still connected. Every three minutes, the Desktop Program sends a **wtbTickleBaseStation** message to the Base Station, expecting a **btwNotDeadYet** reply. If no such message is received within 5 minutes, then the Desktop Program concludes that the connection has been broken.

[55] Grayed-out items are obsolete, retained only for backward compatibility.
[56] Grayed-out items are obsolete, retained only for backward compatibility.

For its part, the Base Station will decide that the connection has been broken after 10 minutes of silence from the Desktop Program. When this happens, the Base Station will send a **btwTimedOut** message on the off chance that it will get through.

When the Desktop Program believes that it has established a USB connection to the Base Station, it introduces itself with a **wtbConnect** message. The Base Station might answer with **btwAlreadyConnected** (suggesting that the Desktop Program is confused), but more often it will say **btwConnected** and then **btwJustConnected.**

When the Desktop Program wishes to break the connection, it sends a **wtbDisconnect** message; before actually closing the connection, the Base Station answers with **btwDisconnectedRQ** and then **btwDisconnectedBS.**

7.2.1.2 Base Station boot-up messages

The Base Station is expected to run continuously, but not to be continuously connected to the Desktop Program. The Desktop Program will therefore usually not be available to receive messages generated during the Base Station's boot sequence. On the off chance that the Desktop Program is listening, the Base Station may send various messages during its boot sequence, either reporting successful progress

btwStartingXBee, btwXBeeSettingDestination, btwXBeeInCommandMode, btwXBeeCommandIssued, btwStartingFileSystem, btwSetupComplete

or reporting difficulty

btwXBeeBufferBusy, btwXBeeDifficultCommandMode, btwXBeeStuckInCommandMode, btwXBeeNoCommandMode, btwFileSystemTrouble, btwXBeeNoSetDestination.

7.2.1.3 directives to the Base Station

The Desktop Program can interrupt a lengthy dump of filed data from the Base Station with a **wtbInterrupt** message; the Base Station's reply is **btwInterrupted**.

With parameters taken from the desktop's own reckoning of UTC time, the **wtbSetTime** message directs the Base Station to set its DS1307 clock. The complementary **wtbReadTime** message directs the Base Station to send back a **btwReadTime** message whose parameters tell the DS1307's time.

A **wtbStartDump** message directs the Base Station to begin dumping the contents of its μSD card.

7.2.1.4 messages passed to specific peripheral stations

As described in Section 8.2 below, a table maintained by the Desktop Program contains the XBee unit identifier (**MY**) of each peripheral board. A message to a specific peripheral board is a tab-delimited string whose components are

wtbSendToXBee <MY> <WTB> <0 or more other parameters>

where <WTB> is one of **wtbRelayAction, wtbRelayHysteresis, wtbRelayLight, wtbRelayMode, wtbRelayOff, wtbRelayOn, wtbRelaySchedule, wtbRelayTemps, wtbTakeVC0706Image, wtbBroadcastPressure,**[57] **wtbBroadcastLocalTime,**

[57] The SCD30 CO_2 sensor needs to take ambient air pressure into account, but air-pressure measurement may not be locally available on the peripheral board carrying the CO_2 sensor. When an air-pressure measurement is received by the Desktop Program, the Desktop Program sends a

wtbSendToXBee <MY> **wtbBroadcastPressure** > <pressure value>

message to each affected peripheral board.

wtbSpeedupChanged. The other parameters are numerical. If for some reason the Base Station is unsuccessful in forwarding the message to the specified peripheral station, then the Base Station sends a **btwFailedTransmission** message back to the Desktop Program.

7.2.1.5 messages passed to annunciators

The **annunciators** module of the Desktop Program hard-codes creation of an annunciator object for each annunciator on the system, specifying what data should be sent to that annunciator. The Desktop Program sends tab-delimited lines like

wtbAnnunciator <MY> <text>

to the Base Station to control the specified annunciator. The <text> is either

- '**Z**' and a numeral specifying the number of lines to expect, or
- A numeral specifying the text size (in [**1 .. 5**]), a numeral specifying the color (**0** or **1**), and the intended text of the line.

7.2.1.6 dumping

Just before the Base Station begins to transmit data from its µSD card to the Desktop Program, it sends a **btwDumping** message. When it is about to pass data through without using the µSD card, it sends **btwPassThrough**. When the Base Station receives a line of data from a peripheral station (formatted as described in Section 5.4.7 above), a timestamp is prepended. If the line is to be saved to the µSD card, then the Base Station sends a **btwSavingBuffer** message after saving each 512-byte block. When the Base Station sends the line to the Desktop Program, it is further prepended with **btwLiveData** if it is being passed through via a live connection, or with **btwDumpData** if it is being dumped from the µSD card.

The files on the Base Station's µSD card are named **1.txt**, **2.txt**, and so on. Almost all the action is in **1.txt**, but **2.txt** is used to hold data that comes in during the dumping of **1.txt**, **3.txt** is used for data that comes in during dumping of **2.txt**, and so on.[58]

When the Base Station receives a line of data from a peripheral station, it writes the board tag[59] to the TeensyView. After 30 seconds with no data received, then the Base Station writes a single period to the TeensyView.

During dumping of µSD data, the Base Station sends a **btwDumpingBlock** message as it prepares to send each 512-byte block to the Desktop Program. When all of the µSD data have been transferred to the Desktop Program, or when dumping has been interrupted,[60] the Base Station sends a **btwDumpEnded** message.

[58] I've seen **3.txt** come into play a few times. I've never seen any higher-numbered file used, but it could happen.

[59] See Section 5.4.5 above.

[60] See Section 7.2.1.3 above.

8 Desktop Program

The Desktop Program (about 43 000 lines of Delphi) runs in Windows. Its top-level form is small,

Figure 5 top-level form, with main menu

and most of its work is done through various other forms that will be described. Closing the main form closes the application; double-clicking any other form sends focus to the main form.

When the Desktop Program is started, the forms displayed in addition to the main form are the **`Dashboard`**,[61] the **`Log`**,[62] and the panels showing current readings of the various sensors.[63] As described in Section 8.10.2 below, some graphs may also be displayed at this time. Clicking the **`arrange`** item at **F** in the top-level menu moves the open forms into a favored arrangement on the primary screen.

When the application is closing, the user is given the option to save into text files various logs that, as described variously below, are maintained in listboxes during operation of the application.

Some parameters used by the Desktop Program are taken from an INI file (**`HC4.INI`**) that is kept in the same directory as the program's **`.exe`** file.

The fields at **H**, **I**, and **J** are provided to alert me to malfunctioning components.

- **H** is used for various messages.
- **I** identifies the peripheral board that has transmitted data least recently. Some sensor-heavy peripheral boards are typically heard from every two or three minutes, but others may speak up only once or twice an hour. Longer silences are suspicious.
- **J** shows how long it's been since any data were received from the Base Station.

8.1 database

The center of the Desktop Program is an Advantage relational database.[64] Most of the tables of the database are used for data received from the peripheral stations, but some tables are used for administrative purposes. Using a general database greatly simplifies modifications and additions to the system. For example, the Weather on a Given Month and Day facility[65] was added with just 177 lines of code.

Some kinds of data have been accumulating since mid-2009, so some of the data tables are large (hundreds of megabytes). The recent data are always of greatest interest, so some data tables are arranged in triples, with one table for recent data, one for older data, and one for the oldest data (typically data from years before the year before last year). As of 2026-07-08, for example, the accumulated temperature/pressure/relative humidity measurements were divided among a 2.8 MB table of recent data, a 32 MB table of older data, and a 307 MB table of data from before 2025. The process of moving data out to

[61] See Section 8.2.1 below.
[62] See Section 8.2.2 below.
[63] See Section 8.6 below.
[64] See https://dbdb.io/db/advantage-database-server (accessed 2021-05-25).
[65] See Section 9.1 below.

the older-data tables is described later in this section. Except for that process, the fact that some tables are triplicated is invisible to the user.

An earlier version of the system time-stamped data only with local time, but Daylight Savings Time made the interface messy. Tabulated data are now stamped with both UTC time (as provided by the Base Station) and local time (for the user interface).

The data tables currently maintained are

name	content
CO2	CO_2 levels (ppm)
Light (triplicated)	Light outside and in living room
OnOff (triplicated)	A/C, furnace, various relays
Pond (triplicated)	pond depth and temperature
Power	current on each side of house power
Rain (triplicated)	rain gauge
SPS30 (triplicated)	particulate-related data
Thermostats	target temperatures of relay-based thermostats
TRHP (triplicated)	air temperature, relative humidity, dew point, pressure
VC0706	file numbers and timestamps
Wind (triplicated)	wind speed & direction

The **CO2, Power,** and **Thermostats** tables are not triplicated because it is not expected that non-recent data of these kinds will be of any interest. The **VC0706** table is not triplicated because new files on the Base Station's μSD card overwrite old ones, and the list of files is therefore of bounded size.

Each record in the **CO2, Light, OnOff, SPS30**, **Thermostats**, **TRHP,** and **VC0706** tables has a field to indicate what room or area is the subject of the measurement. No such fields are used in the other tables, where the targets of observation are implicit.

When an air temperature and relative humidity have been provided by a peripheral station, the dewpoint and Humidex are calculated for storage in the **TRHP** table.

8.1.1 non-contributory data

Data for the **Light, OnOff, Rain, Thermostats**, **TRHP,** and **Wind** tables might be acquired frequently, but the acquired values are often scarcely changed from reading to reading. For example, the furnace is unlikely to change state (that is, to turn on or turn off) more than a few times a day. Data are added to these tables only to begin a calendar day or when the data show a change. In the implementation, each *unique* datum in some tables appears twice — once when it first appears and again just before it is superseded.

Even when the measured data are changing, the changes may be so small that the new data shouldn't be stored unless the stored data are becoming sparse. For example, a reported air temperature is stored only if (a) it is at least 0.3 °F different from the previous stored value, or (b) at least 24 minutes has passed since the previous stored value was reported. The arbitrary parameters here are those in the current **HC4.INI** file.

8.2 input processing

8.2.1 Dashboard

The initial appearance of the **Dashboard** is

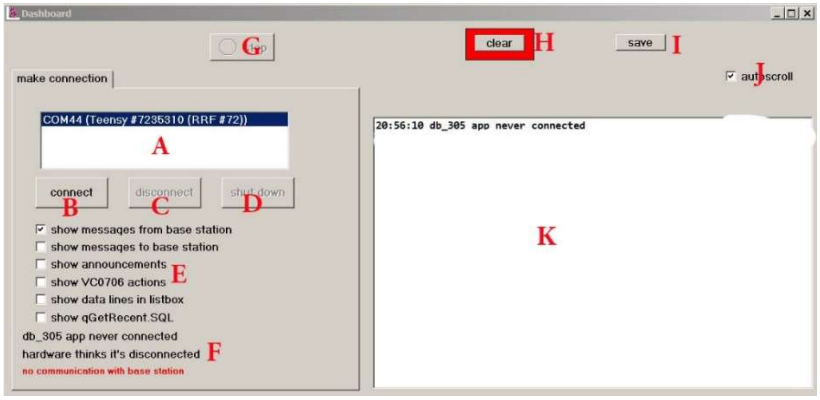


Figure 6 initial appearance of the Dashboard

The listbox at **K** is essentially empty when the application is opened, but it will be filled with a log of application events. The lines of the listbox can be cleared (**H**) or saved to a text file (**I**) in a date-named folder.

The checkboxes at **E** cause the listing at **K** to be more or less verbose, including (or not)

- lines describing lines sent to annunciators,
- messages sent to or from the Base Station,
- a line for each image captured by a VC0706 camera,
- each data line received from any peripheral station,
- certain complex SQL statements that are dynamically created elsewhere in the application.

Available USB ports are listed at **A**; here there was only one. Clicking the button at **B** causes the application to attempt to make a connection to the USB port selected at **A**; the progress of establishing the connection is shown at **K** and its status is shown at **F**.

Once a connection has been established, the appearance of the Dashboard changes:

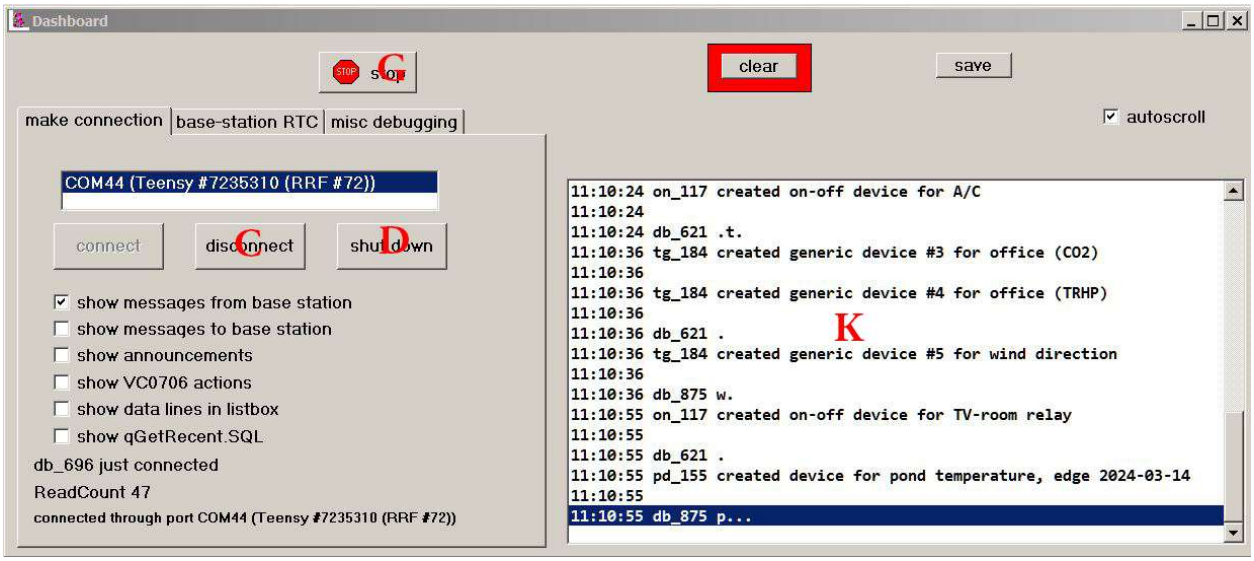


Figure 7 Dashboard after connection to Base Station

The buttons at **C** and **D** are enabled, and they tell the Base Station to break off the connection and to shut down, respectively. When a connection is first established, any data waiting at the Base Station are dumped to the Desktop Program. The dump can be aborted (but data will be lost) by clicking the button at **G**.

During each run of the application, a line like the **`created . . . device . . .`** lines seen here at **K** appears when the application first receives data from a specific sensor. The third-from-last line shown here, for example, reflects the first appearance (during this run of the application) of data from the DHT22 in the pond. The devices created are all descendant instances of the **`tSensorDevice`** object. Creation of the instance draws on configuration tables described in Section 8.3 below, so that the instance's methods can store the incoming data in the appropriate fields of the appropriate tables.

Lines similar to the last line here are generated as data continue to arrive. A dot is displayed for each line whose data is stored in the database. If the line was determined to be non-contributory,[66] then a lower-case letter is displayed: **`p`** for discarded pressure values, **`w`** for discarded wind data, and so on.

The second page of the **`Dashboard`** is used for control of the DS1307 clocks. They allow the Base Station's clock to be read and set, and for the local time to be broadcast as a new setting for all of the peripheral boards.

The third page of the **`Dashboard`** provides some rarely-used debugging tools, and it allows the **`HC4.INI`** file to be displayed and edited.

The **`Dashboard`** form can be closed, and it can later be reopened by selecting **`dashboard`** from the main menu.

[66] See Section 8.1.1 above.

8.2.2 Log

The center of the **Log** form is a listbox **L** that can record, and lightly decipher, each line of data received from the Base Station. A typical appearance is

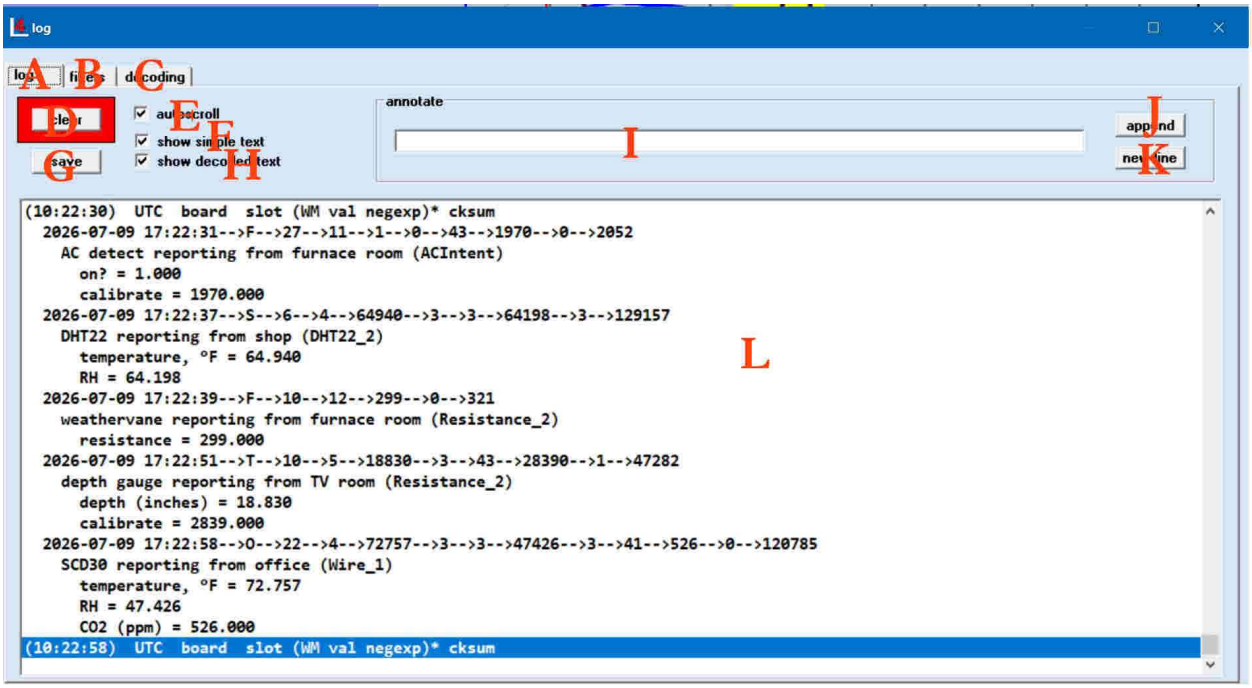


Figure 8 log

Because the boxes at **F** and **H** are here both checked, each received line is reported in two forms. Each of the "simple text" lines consists of the Base Station's UTC timestamp, the station tag, the board slot tag, the triple(s), and the checksum. The immediately following "decoded" lines decipher the simple line.

The lines of the listbox can be cleared or saved to a text file in a date-named folder via the buttons at **D** and **G**, respectively. Text entered at **I** can be inserted into **L**, either extending the current line (**J**) or as a new line after the current line (**K**).

The tab at **B** contains another listbox and controls that allow selectivity (by board, area observed, or sensor type) of the data lines listed. The tab at **C** just shows a reference list of the WhatMeasured[67] codes.

The **Log** form can be closed, and it can later be reopened by selecting **view/log** from the main menu.

[67] See Section 5.4.2

8.3 Sensor Manager

Clicking **sensors/sensor manager** on the main menu evokes the Sensor Manager form.

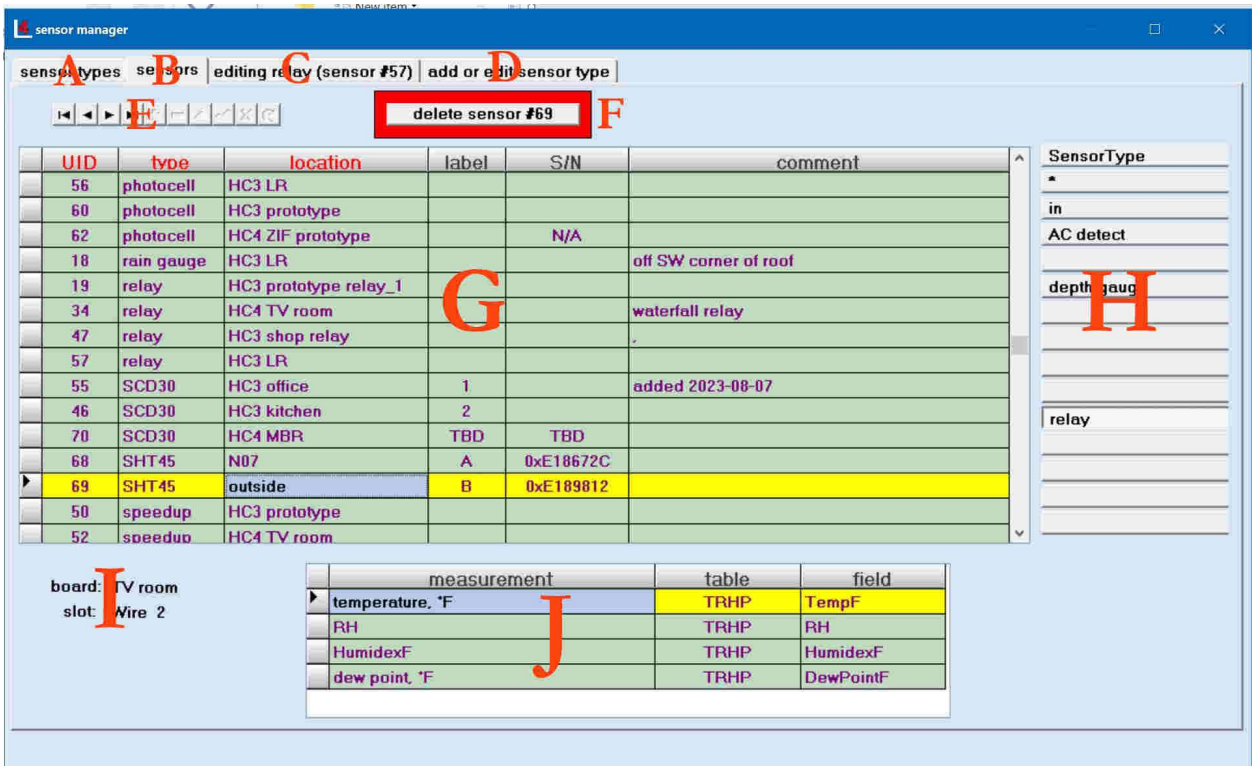


Figure 9, Sensor Manager

At **G** on the principal page of this form there is a list of all of my sensors, whether or not they are in use. The grid at **J** shows how data from the sensor selected at **G** are stored in the database. Double-clicking an entry at **G** allows (using the tab at **C**) information about the selected sensor to be edited: what it observes, to what slot (if any) of what board (if any) it is connected, how its data are to be stored in the database, and so on. The recognized sensor types are listed at tab **A**, and the list can be edited or expanded at tab **D**.

This figure shows two conventions:

- If the color of the text in a grid's column header is red, then clicking on that header sorts the records of the grid into ascending order by the entries of that column.
- A field shown with purple text on a green background (here, all of them) is read-only.

The index at **H** allows rapid access to records in the grid at **G**.

8.4 Board Manager

Clicking **sensors/board manager** on the main menu evokes the Board Manager form.

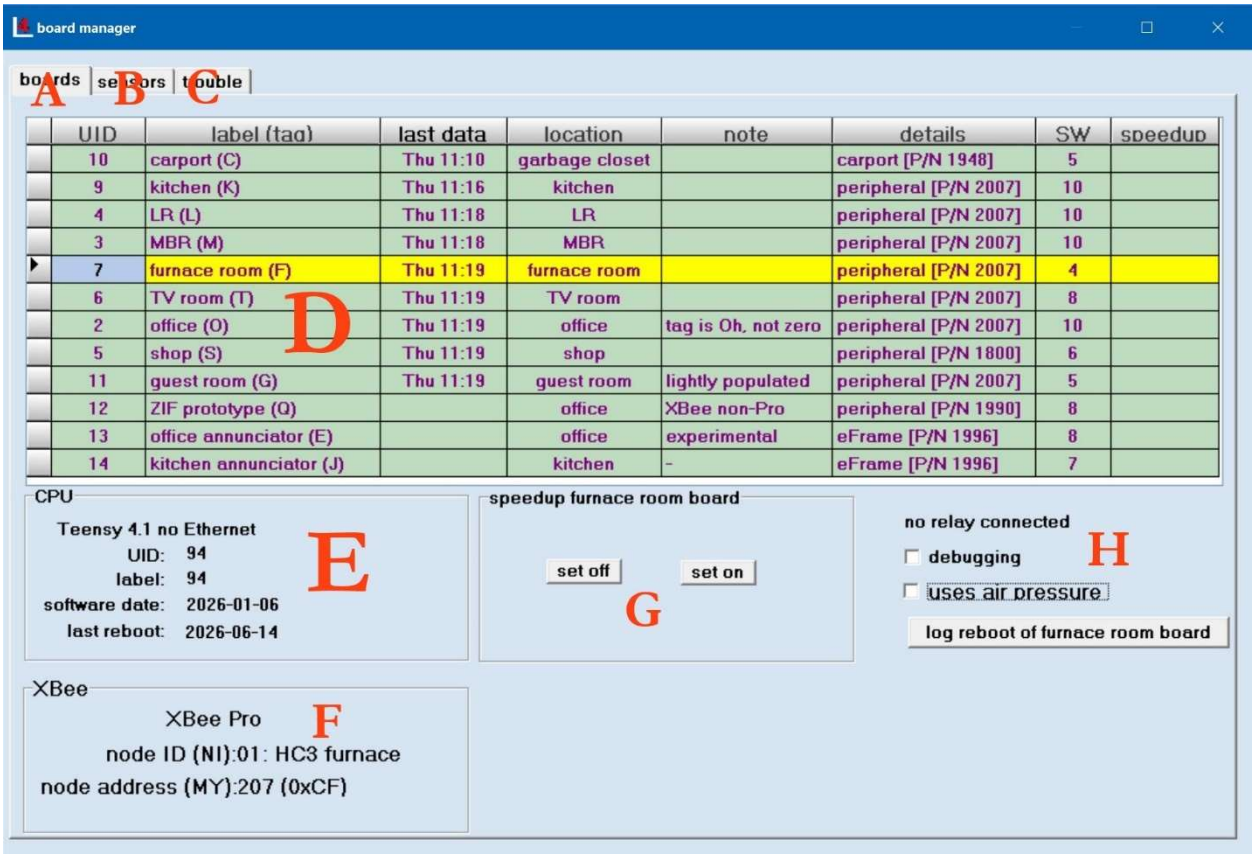


Figure 10, Board Manager

The grid at **D** lists all of the boards of the system, with additional board-specific information below.

As noted in Section 5.4.8 above, a board may be directed to speed up its reading and reporting by a factor of approximately 10. This feature is controlled by the buttons at **G**, which cause transmission of **wtbSpeedupChanged** messages to the selected board.

As noted in footnote 57, the CO_2 sensors try to take account of ambient air pressure. When the Desktop Program receives a pressure reading, it transmits the value to each of the boards here recorded (**H**) as needing it.

Because peripheral stations and annunciators are added to the system only rarely, the main table behind this display (**Boards**) is maintained by hand, using the Advantage **arc32** tool.[68]

The tab at **B** provides a list of all of the sensors attached to the selected board, and allows the user to specify to which of the board's connectors it is attached. The tab at **C** carries a list of specified but impossible connections.

[68] See https://devzone.advantagedatabase.com/dz/content.aspx?Key=20&Release=16&Product=8&Platform=6 (accessed 2021-05-25).

8.5 relay control

Clicking **`view/relays`** on the main menu brings up the relays form, which initially looks like this:

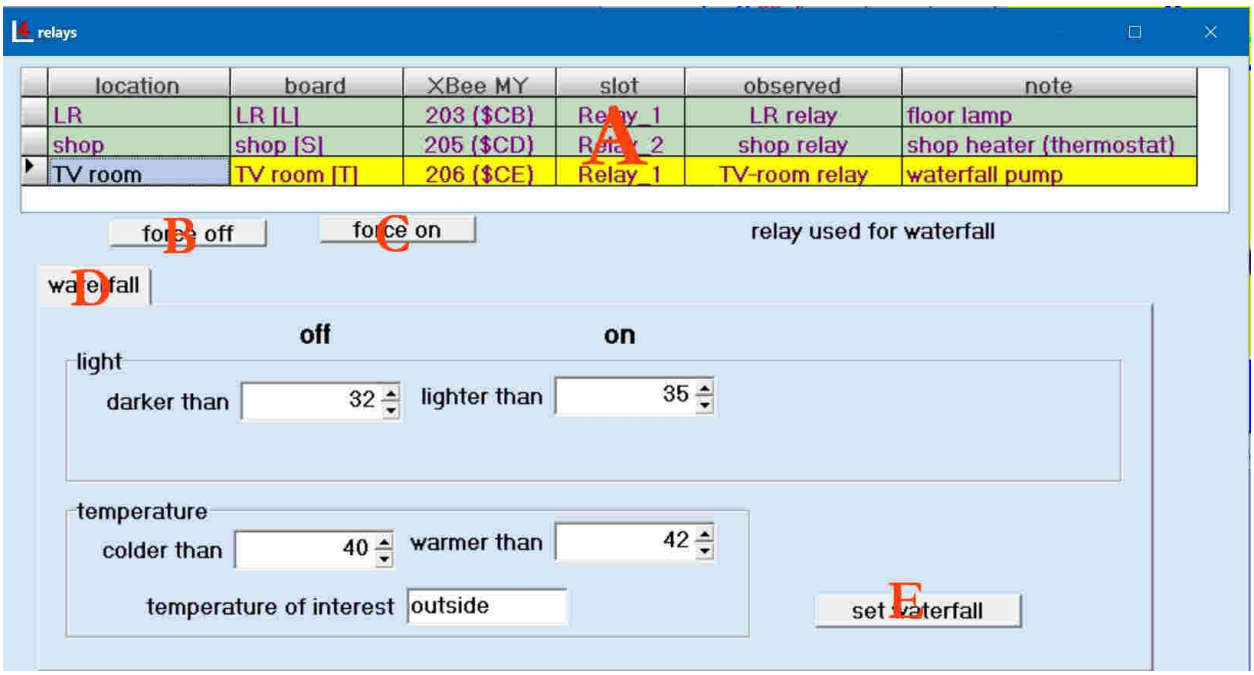


Figure 11, relay form, initial state

The peripheral stations that have relays are listed at **A**. The buttons at **B** and **C** can be used to control the relay from the Desktop Program, overriding peripheral control.

The component at **D** is a tabbed control, but only the tab pertinent to the selected relay is shown. When the parameters needed by the selected relay's driver are chosen,[69] the button at **E** causes them to be transmitted to the selected relay's board, using some combination of the tags **`wtbRelayAction`**, **`wtbRelayHysteresis`**, **`wtbRelayLight`**, **`wtbRelayMode`**, **`wtbRelayOff`**, **`wtbRelayOn`**, **`wtbRelaySchedule`**, and **`wtbRelayTemps`**.

8.6 VC0706 images

As noted above, the bulky images captured by VC0706 cameras are saved on the peripheral-stations' µSD cards instead of being transmitted to the Base Station. The base station is notified of each image's

[69] A convention here demonstrated is that fields shown with black text on a white background are editable.

creation, and the notifications are saved in the **VC0706** table. Clicking **view/image list** on the main menu brings up this form:

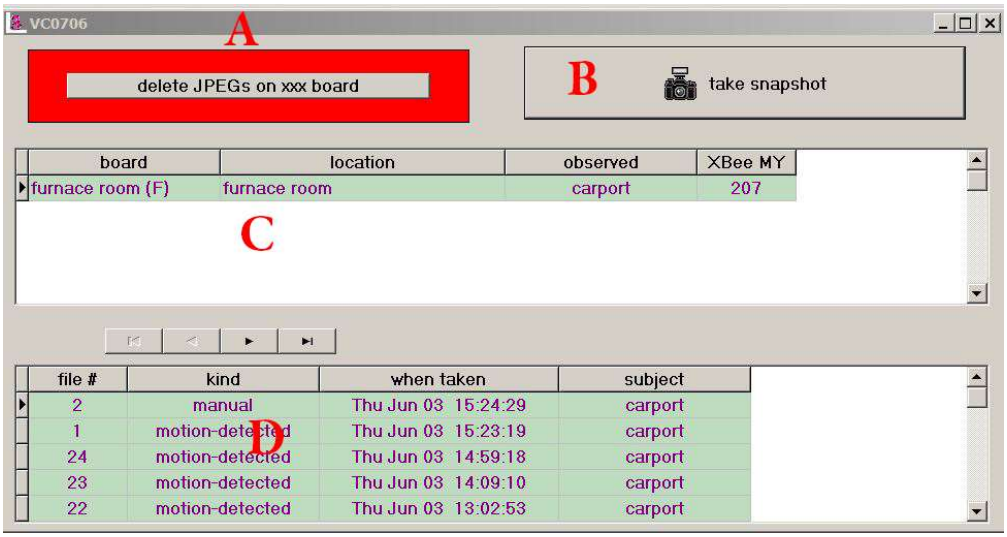


Figure 12, VC0706 image

The grid at **C** lists all of the peripheral stations that have VC0706 cameras connected to them (here, there is just one). The button at **A** directs the selected station to delete the image files on its µSD card; the button at **B** directs the station to snap a picture. The grid at **D** lists all of the images thought to be stored on the selected station's µSD card.

8.7 sensor-display forms

Two passive forms display the latest values of data arriving from the peripheral stations, one for the outdoor sensors

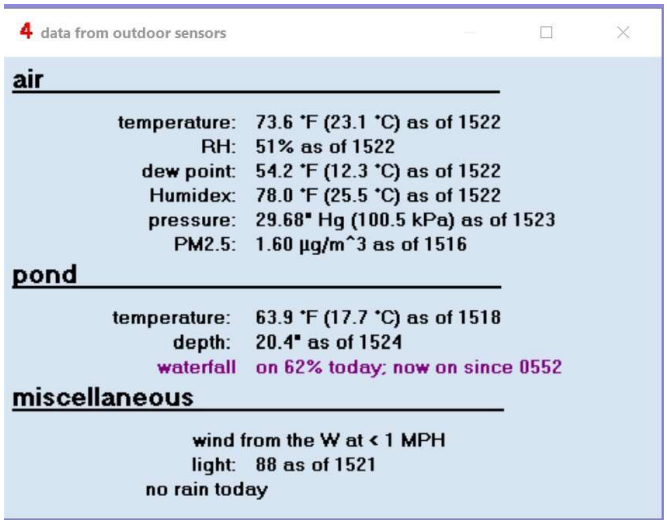


Figure 13, outdoor sensors

and one for the indoor sensors:

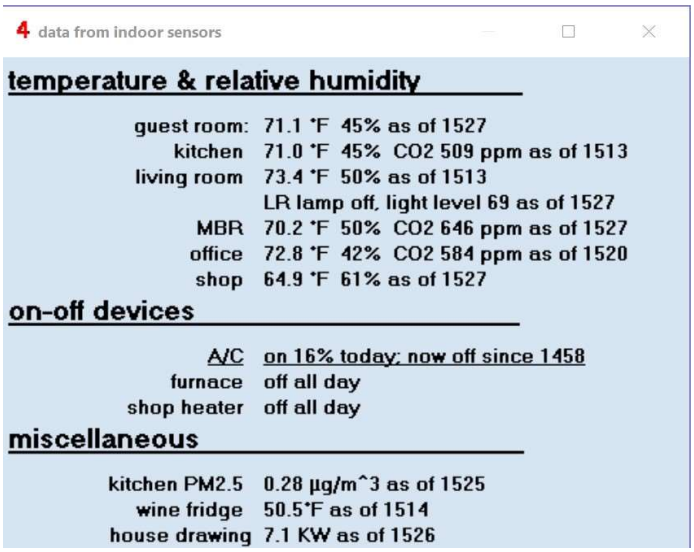


Figure 14, indoor sensors

These forms have only limited access to the database, so when the application opens, most of the entries in these forms are greyed out, and entries become active one by one as data come in.

These forms can be closed, and they can later be reopened through the main-menu items **`sensors/outdoor`** and **`sensors/indoor`**.

8.8 the **`annunciators`** module

As data are provided to the sensor-display forms, the data are selectively passed on to the annunciators module, which in turn organizes the text lines displayed on the annunciators. When one of the annunciators has not recently been updated, and all of the data it expects are available, the annunciator module formats lines containing the current data and transmits those lines to the affected annunciator.

8.9 extremes

A form allowing one to explore the extremes of the collected data is available from **`view/extremes`** on the main menu.

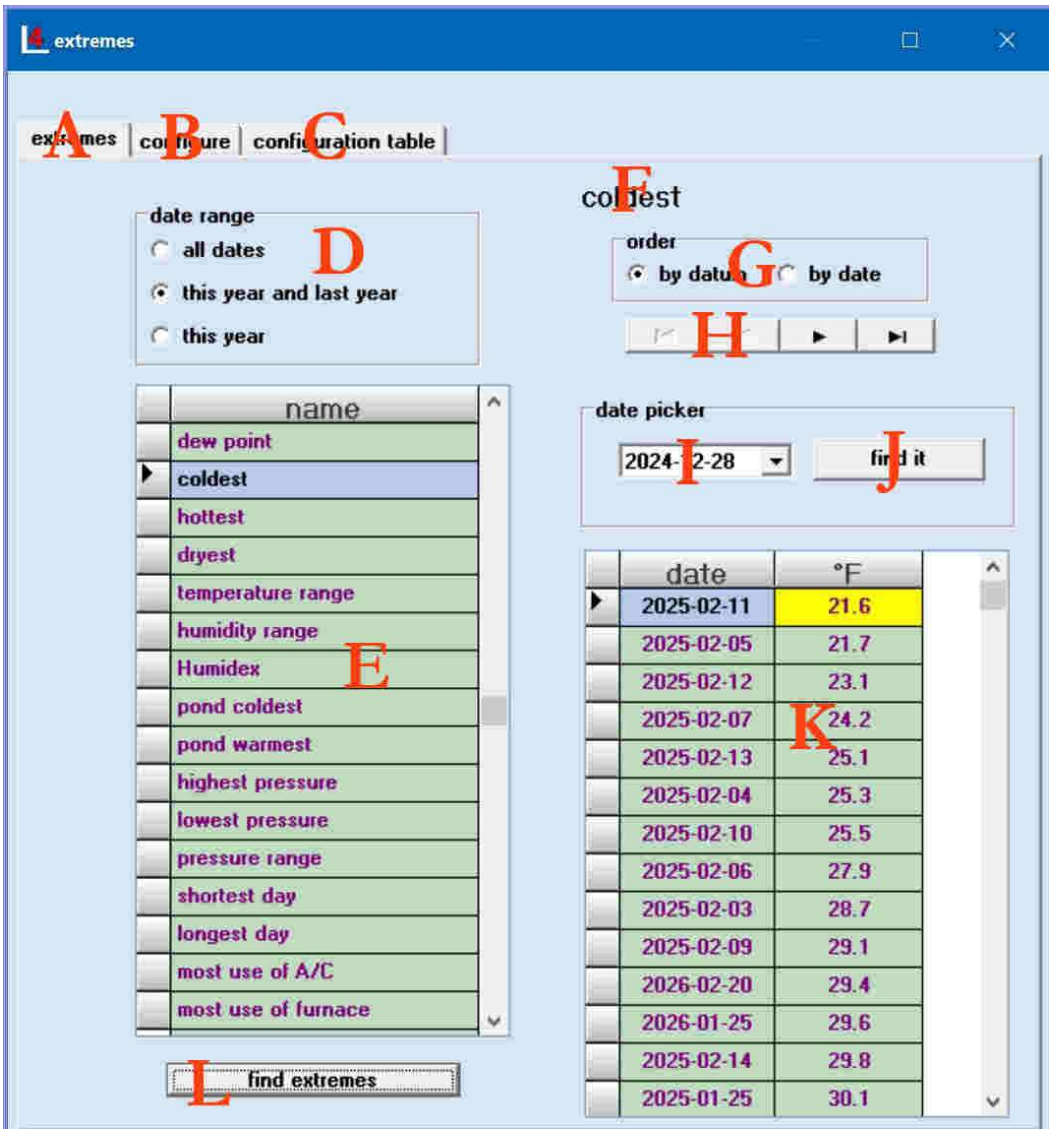


Figure 15, extremes

When this form is opened, the only visible controls are the radio buttons at **D**, the grid at **E**, and the button at **L**. When a date range (**D**) and a measurement **C** are chosen, pushing the **L** button causes the other controls (**F-K**) to become visible. The radio buttons at **G** cause the records shown at **K** to be ordered in descending order by date, or in descending order by extremeness. The button at **J** moves focus in the **K** grid to the record with the date selected at **I**.

The **B** and **C** tabs provide facilities for creating options to be added to the grid at **E**.

8.10 graphing

The sub-entries under **graphs** on the main menu provide access to predefined graphs, user-defined graphs, and a tool (the Graph Manager) for the creation and editing of user-defined graphs. The graphs get their data from the database, and they are automatically replotted when new pertinent data appear while they are visible. To avoid thrashing, new data are not allowed to trigger replotting of a given graph more often than every *n* seconds, where *n* is taken from the **HC4.INI** file.

8.10.1 pre-defined graphs

The system includes three graphs whose design requires options not available to user-defined graphs. The "grouped outside" graph (accessed via **graphs/outdoor/outside, grouped by date** on the main menu) pulls together various data, notably the photoperiod and temperature range.

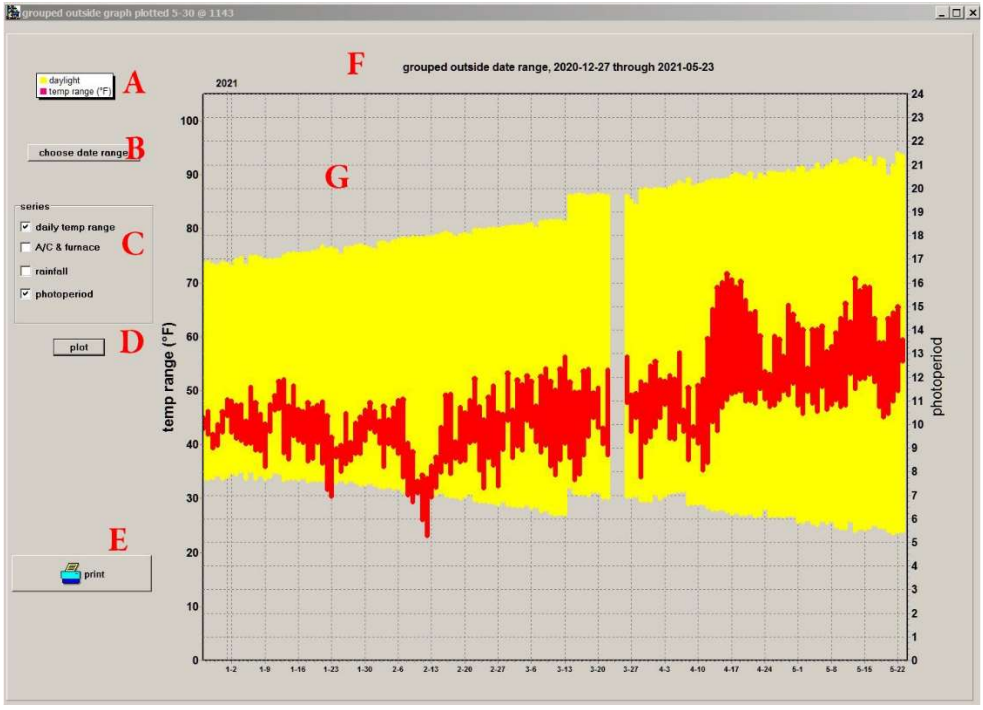


Figure 16 graph: grouped outside

The button at **B** brought up a date-range dialogue, and the boxes at **C** allowed the data displayed to be limited to the temperature range and sunrise/sunset. In this example, the photoperiod jogged (in local time) as Daylight Savings Time began, and there was a gap corresponding to a few days in March 2021 when the system was down.

Particulate-related data from the outdoor SPS30 can be part of a user-defined graph, but I threw the $PM_{2.5}$ data into the Wind graph (accessed via **`graphs/outdoor/wind & particulates`** on the main menu).

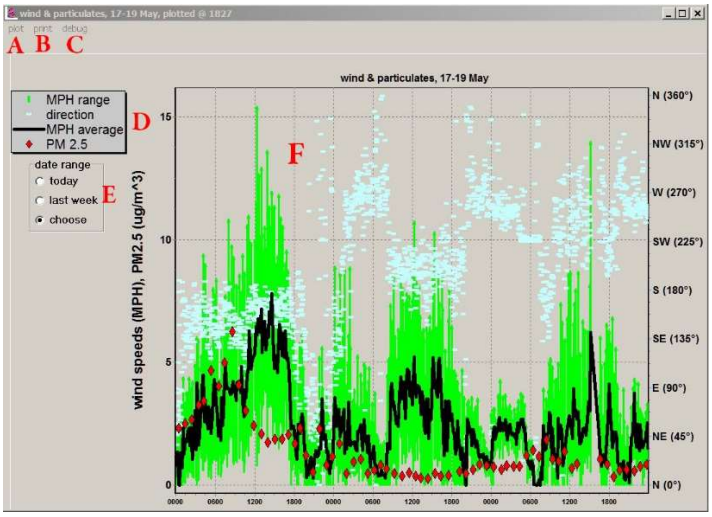


Figure 17, graph: wind

In the Wind graph, some controls have been moved into a menu (**A/B/C**), and the common date-range choices (**`today`** and **`last week`**) at **E** allow the range-picking dialog to be avoided. Some features of the Wind graph (the specialized right axis, the ranges) are not available in user-defined graphs.

The **`outdoor/annualized temperature ranges`** graph draws on the **`Summary`** table to show how this year's daily temperature ranges compare to those of previous years:

Figure 18, temperature ranges

8.10.2 user-defined graphs

Here is a typical user-defined graph, demonstrating many of the options available:

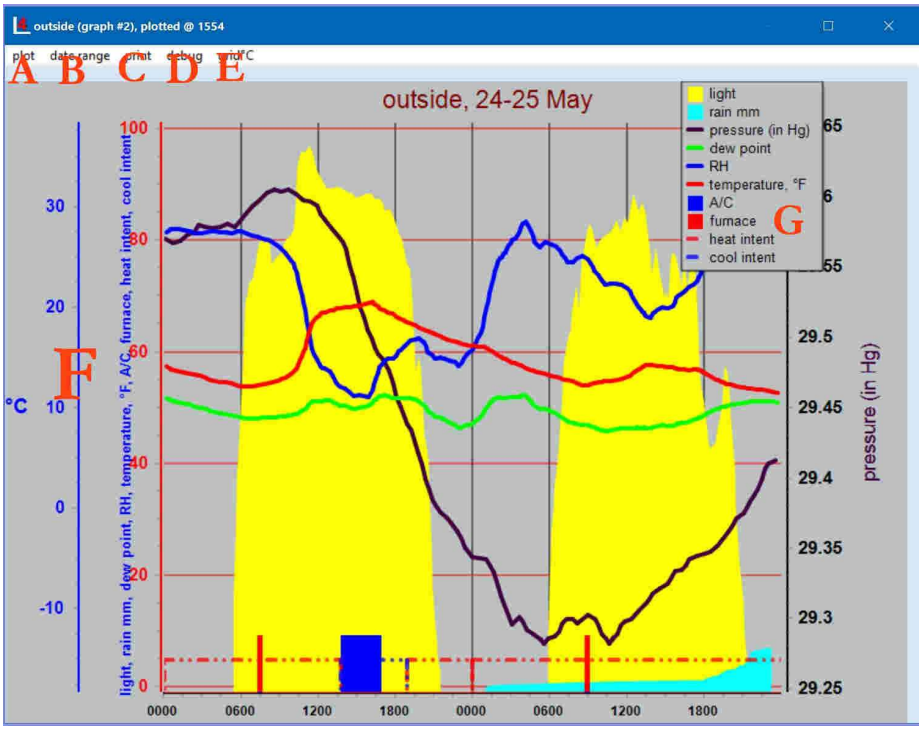


Figure 19, user-defined graph

After this graph had been defined and named **`outside`**, by tools described below, it was accessed by clicking **`graphs/frequent/outside`** on the main menu. It initially appeared showing only data from the current date, as is the default behavior for all the graphs, but it was redrawn after a date range of interest was selected through a dialog accessed at **B**.

Some parts of the user-defined graph system are hard-coded. This sample graph contains

- **line plots** (temperature, relative humidity, Humidex, and pressure),
- **area plots** (furnace, A/C, light, and rain), and
- **dash-dot-dot plots** (heat intent, cool intent).

A **point plot** ($PM_{2.5}$) is visible above in Figure 17. In general, each **`tWhatMeasured2`**[70] value is permanently assigned a plot type:

- **`wm2DepthInches, wm2EventCount`**[71], **`wm2IsNowOn`**, and **`wm2Resistance`**[72] are associated with area plots
- **`wm2MassPM25`** and **`wm2Voltage`** are associated with point plots; and
- all other measurements are associated with line plots, except that as described below, a plot that would otherwise be an area plot can be manually selected to appear as a dash-dot-dot plot.

[70] See Section 5.4.2 above.

[71] This association optimizes the graphing of accumulated rain.

[72] This association optimizes the graphing of light. The wind-direction data also arrive as resistance measurements, but they are, as noted in the previous section, handled separately.

Also, every user-defined graph has the same menu (**A B C D E**). The format and location of the legend at **G** are not under user control, nor is the calibration of the X-axis.

Clicking the menu item at **E** interchanges the two scales at **F** and causes the horizontal grid lines in the graph to reflect the then-inner of the two scales.

8.10.2.1 Graph Manager

Everything else is user-specified, via the Graph Manager that is accessed via **`graphs/graph manager`** on the main menu.

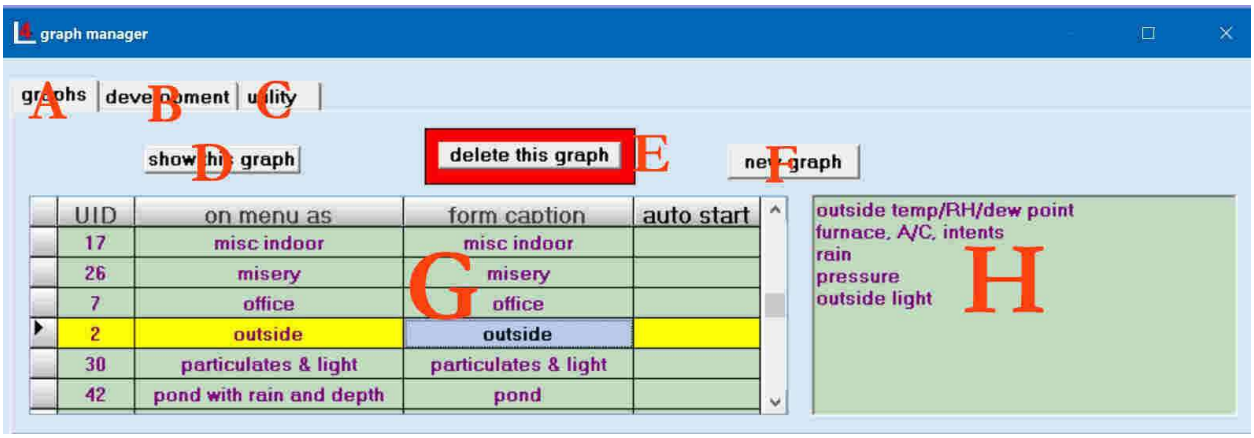


Figure 20, Graph Manager

The Graph Manager presents (at **G**) a list of all the existing graphs, with a description of the selected graph at **H**. The work of editing an existing graph (by double-clicking at **G**) or creating a new one (**F**) is done in the Graph Editor.

8.10.2.2 Graph Editor

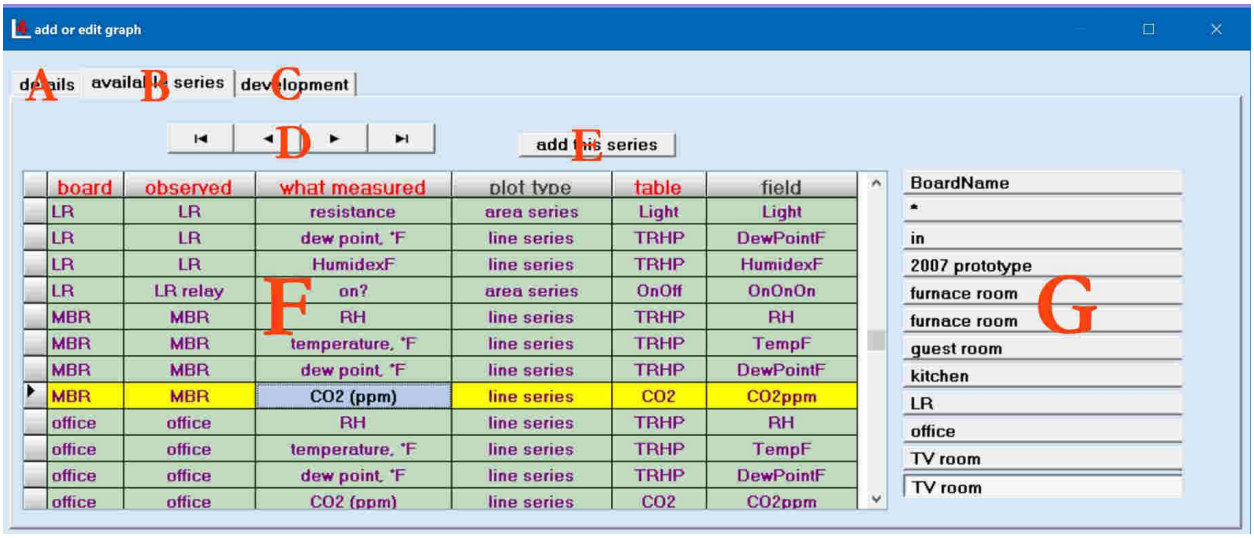


Figure 21, graph editor

The most fundamental work of using the graph editor consists of selecting series to add to a new or existing graph. The **B** tab shown here displays a list of all the available series at **F**. The index at **G** facilitates navigation of the long grid at **F**. The button at **E** adds the selected series to the graph being modified.

The **details** tab here at **A** allows the user to specify how the selected series are to be presented. The **details** tab has three subtabs. The global tab at **H** looks like this:

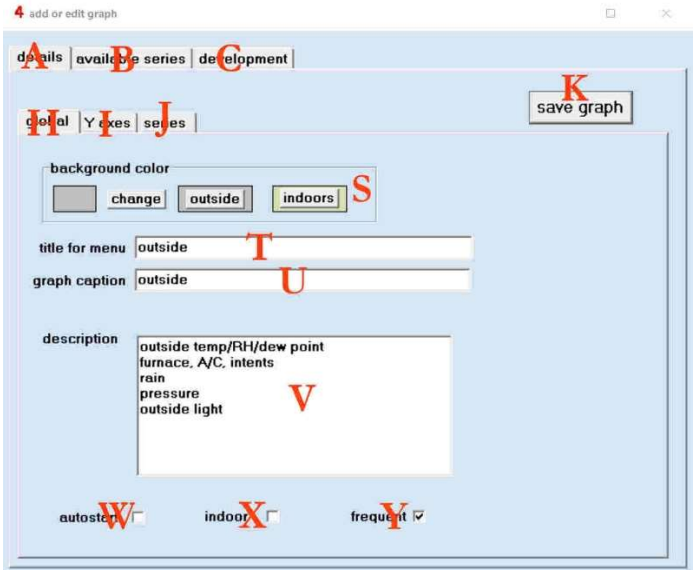


Figure 22, graph global details

At **S**, this form determines the background color of the graph. There are default colors for graphs of **indoors** and **outside** data, but an arbitrary background color can be chosen via the **change** button.

The edit boxes at **T** and **U** and the memo box at **V** determine the text used as a title on the graph, the text that appears on the submenu below the top-level **graphs**, and the optional description that appears on the main graph manager form (**H** in Figure 20), respectively.

If the checkbox at **W** is checked, the graph will be displayed when the application is started.

When the available graphs are listed in the submenu below **graphs**, they are grouped as **frequent** (if the box at **Y** is checked), **indoor** (if the box at **X** is checked), or **outdoor**.

The **`Y axes`** tab at **I** allows the graph's left and right Y axes to be defined.

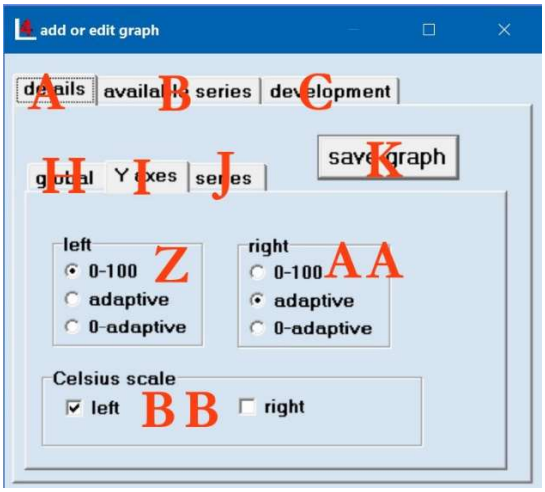


Figure 23, graph Y axes

The scale of each axis can be set (**Z**, **AA**) to run from 0 to 100, to have a range just greater than the range of the graph's data (recomputed each time the graph is plotted), or with zero as the low end of the range, with the upper end adapted to the data. If one (or both) of the checkboxes at **BB** is checked, then a second scale is displayed for the selected axis, showing Celsius calibrations assuming that the primary scale is degrees Fahrenheit.

Most of the work of the graph editor is done in the **`series`** tab (**J**).

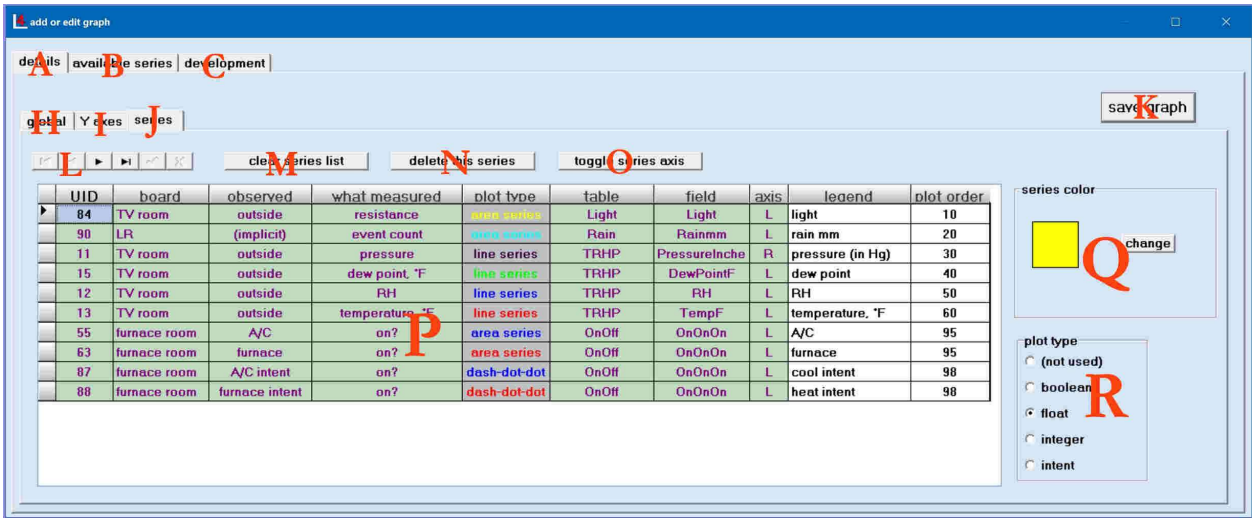


Figure 24, graph editor, series presentation

For each series in the **P** grid, the user must choose

- which scale (that at the left side of the graph, or that at the right side) will apply to it (**O**);
- its font color (**Q**);
- the text that will identify it in the graph legend; and

- the order in which it is plotted. Plot order comes into play when series overlap, because later-plotted series overwrite the pixels of earlier-plotted ones. In Figure 19, for example, the big yellow light series was evidently plotted first, since every other series overwrites it.

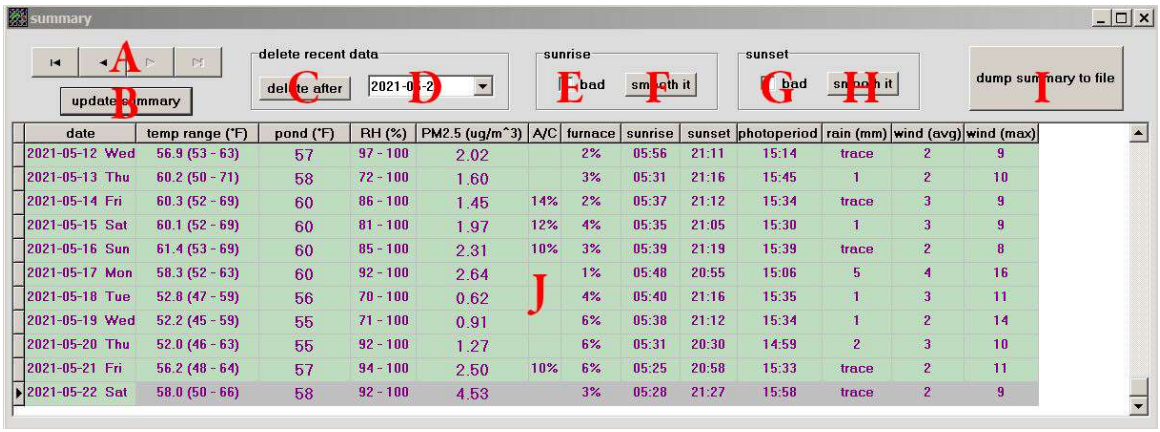
Also, the radio group at **R** allows the plot type of the selected series to be changed. The only use of this now is to recast the "intent" values (saved with **wm2IsNowOn**, and otherwise fated to appear as area plots) to be displayed as dash-dot-dot (Boolean) plots.

Once all of the details are set, the **save graph** button at **K** adds the graph to the collection. A new graph will not appear in the **graphs** menu until the application is restarted.

9 miscellaneous

9.1 summary

Clicking **view/summary** on the main menu brings up the summary form. At the center of this form, a grid displays the **Summary** table, which has one record for every day of recorded data. As of 2026-07-09, my table contains 6008 records.



date	temp range (°F)	pond (°F)	RH (%)	PM2.5 (ug/m^3)	A/C	furnace	sunrise	sunset	photoperiod	rain (mm)	wind (avg)	wind (max)
2021-05-12 Wed	56.9 (53 - 63)	57	97 - 100	2.02		2%	05:56	21:11	15:14	trace	2	9
2021-05-13 Thu	60.2 (50 - 71)	58	72 - 100	1.60		3%	05:31	21:16	15:45	1	2	10
2021-05-14 Fri	60.3 (52 - 69)	60	86 - 100	1.45	14%	2%	05:37	21:12	15:34	trace	3	9
2021-05-15 Sat	60.1 (52 - 69)	60	81 - 100	1.97	12%	4%	05:35	21:05	15:30	1	3	9
2021-05-16 Sun	61.4 (53 - 69)	60	85 - 100	2.31	10%	3%	05:39	21:19	15:39	trace	2	8
2021-05-17 Mon	58.3 (52 - 63)	60	92 - 100	2.64		1%	05:48	20:55	15:06	5	4	16
2021-05-18 Tue	52.8 (47 - 59)	56	70 - 100	0.62		4%	05:40	21:16	15:35	1	3	11
2021-05-19 Wed	52.2 (45 - 59)	55	71 - 100	0.91		6%	05:38	21:12	15:34	1	2	14
2021-05-20 Thu	52.0 (46 - 63)	55	92 - 100	1.27		6%	05:31	20:30	14:59	2	3	10
2021-05-21 Fri	56.2 (48 - 64)	57	94 - 100	2.50	10%	6%	05:25	20:58	15:33	trace	2	11
2021-05-22 Sat	58.0 (50 - 66)	58	92 - 100	4.53		3%	05:28	21:27	15:58	trace	2	9

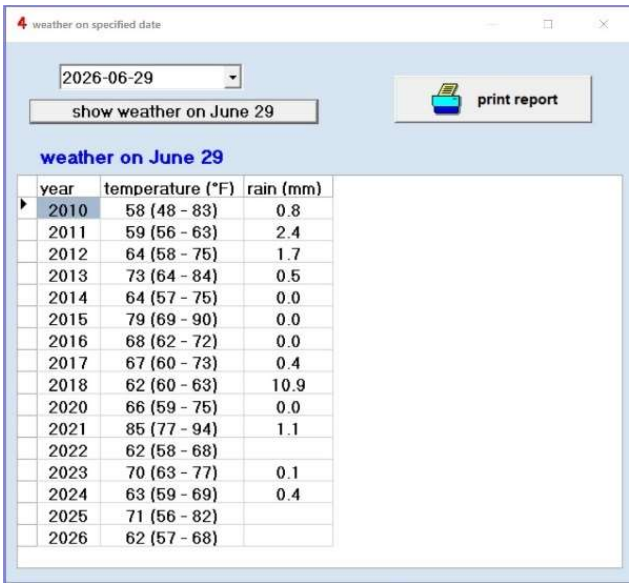
Figure 25, summary

The button at **B** gathers into the **Summary** table any data in the various data tables that are from days later than the last date already recorded here. If data have been corrupted, pulled into the **Summary** table, and then corrected in the data tables, one can clean up the **Summary** table by deleting recent records (**C** and **D**) and then re-updating (**B**).

When a time in the **sunrise** column is thought to be bad, it can be marked as such by checking the box at **E**. Then, as soon as this record is no longer the last one, the implausible sunrise time can be replaced (button **F**) by the average of the entries of the previous and following days. The box and button at **G** and **H** perform similarly for entries in the **sunset** column.

9.2 weather on specified month and day

The **utilities/weather on given date** menu item brings up a form that allows the user to specify a month and day to see the temperature range and rainfall seen on that day in years past. I like to send a copy of this output to people planning to visit us from out of town.

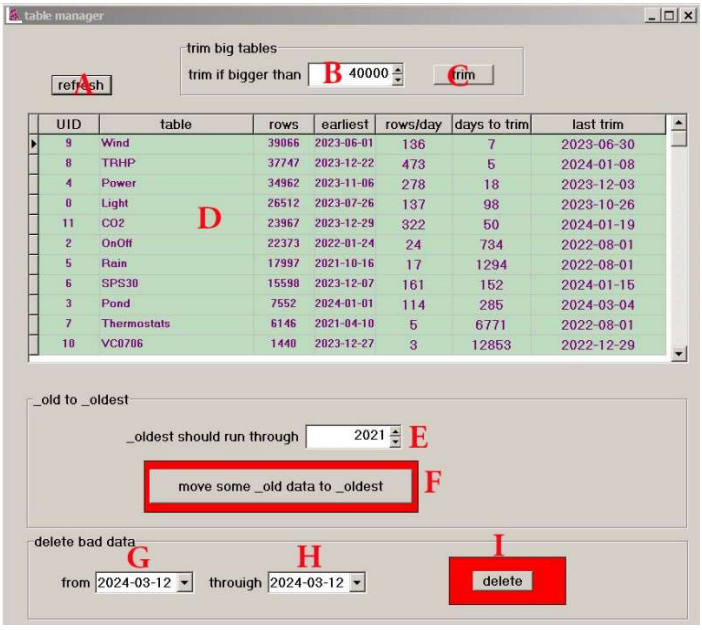


year	temperature (°F)	rain (mm)
2010	58 (48 - 83)	0.8
2011	59 (56 - 63)	2.4
2012	64 (58 - 75)	1.7
2013	73 (64 - 84)	0.5
2014	64 (57 - 75)	0.0
2015	79 (69 - 90)	0.0
2016	68 (62 - 72)	0.0
2017	67 (60 - 73)	0.4
2018	62 (60 - 63)	10.9
2020	66 (59 - 75)	0.0
2021	85 (77 - 94)	1.1
2022	62 (58 - 68)	
2023	70 (63 - 77)	0.1
2024	63 (59 - 69)	0.4
2025	71 (56 - 82)	
2026	62 (57 - 68)	

Figure 26 weather on specified month and day

9.3 Table Manager

The Table Manager form, invoked from the main menu with **`utilities/table manager`**, allows data to be removed from a table of recent data, either moved to the paired table of older data or just discarded. Middle-aged data ("_old") can also be moved out to the tables of oldest ("_oldest") data.



UID	table	rows	earliest	rows/day	days to trim	last trim
9	Wind	39066	2023-06-01	136	7	2023-06-30
8	TRHP	37747	2023-12-22	473	5	2024-01-08
4	Power	34962	2023-11-06	278	18	2023-12-03
0	Light	26512	2023-07-26	137	98	2023-10-26
11	CO2	23967	2023-12-29	322	50	2024-01-19
2	OnOff	22373	2022-01-24	24	734	2022-08-01
5	Rain	17997	2021-10-16	17	1294	2022-08-01
6	SPS30	15598	2023-12-07	161	152	2024-01-15
3	Pond	7552	2024-01-01	114	285	2024-03-04
7	Thermostats	6146	2021-04-10	5	6771	2022-08-01
10	VC0706	1440	2023-12-27	3	12853	2022-12-29

Figure 27, table manager

The button at **C** disposes of about 80% of the rows of each current-data table that has more than the number of rows specified at **B**.

More radically, if a table is thought to contain bad data, those data can be deleted (**I**) as specified at **G** and **H**.

9.4 furnace/AC filter

If I inform the application whenever I change the filter in my house's HVAC system (**utilities/reset furnace filter** on the main menu), then the application can tell me (**utilities/furnace filter**) how many hours of use this filter has had.

9.5 notes

The **view/notes** item in the main menu gives access to a categorical system of notes

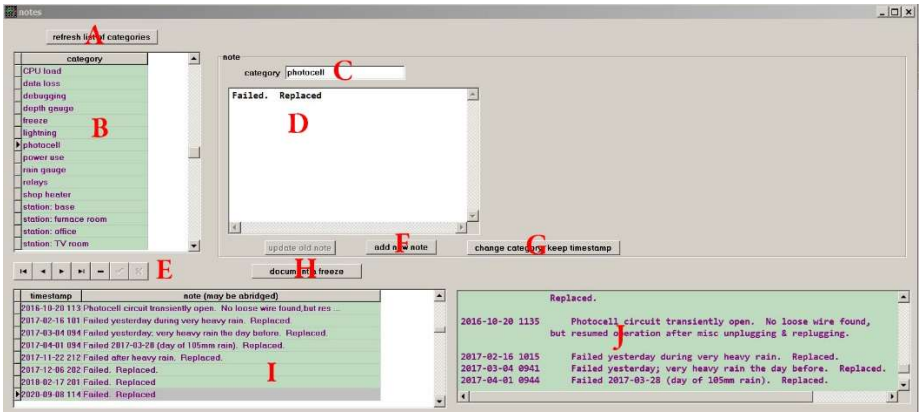


Figure 28, notes

I have used this form for a variety of note-taking, including component monitoring, software to-do lists, and so on. New categories can be created, and old notes and categories can be deleted as needed.

9.6 printer selection

The main-menu **utilities/select the printer** item lets the user choose the printer that will be used when the **print** item is selected on one of the graphs.

9.7 clear warnings

A message becomes visible on the main form (**H** in Figure 5) whenever a VC0706 camera at a peripheral station has detected motion, or a peripheral station reboots. If the application thinks that it is connected to the Base Station, but it has not heard from the Base Station for an unreasonably long time, then the label in the lower right of the main form (**J** in Figure 5) is garishly recolored, and the computer beeps. These alerts are reset by the **tweaks/clear warnings** item on the main menu.