`

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 seven**), annunciators** (now just one), and a special-purpose **Carport Station**.

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

Each of the my older peripheral PC boards carries a small relay and various sensor-specific hardware. In the newer boards, these task-specific components have been exiled to satellite boards, to be deployed and attached as needed.

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 have always been socketed onto my project's PCBs.

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 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.

1.1.5 annunciator display

Each annunciator uses an Adafruit eInk display.[1] These displays have only a limited service life, as counted in text updates. In hindsight, other displays would have been better.

1.2 peripheral stations[2]

(Each of my older peripheral boards was built around a Teensy 3.5. Those boards are not described here.)

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,[3] 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, from batteries, or from the Teensy's USB connector;
- a battery-backed DS1307 calendar clock;[4]
- an XBee radio transceiver;[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. In addition to the (ill-chosen) Adafruit eInk display mentioned above, each annunciator also includes a Teensy 3.2[7] and an XBee transceiver.

[1] See https://www.adafruit.com/product/4947 (accessed 2021-08-30).
[2] See Section 5 for details.
[3] See https://www.pjrc.com/store/teensy41.html (accessed 2024-03-12).
[4] See https://datasheets.maximintegrated.com/en/ds/DS1307.pdf (accessed 2021-05-25).
[5] See https://www.digi.com/xbee (accessed 2021-05-25).
[6] See Section 7 for details.
[7] See https://www.pjrc.com/store/teensy32.html (accessed 2021-08-30).

The eInk display does not tolerate being updated more often than every few minutes. Subject to this restriction, an annunciator can be used to provide near-real-time reports on data relayed from the peripheral stations by the Base Station. A screen-grab from the one existing annunciator looks like this

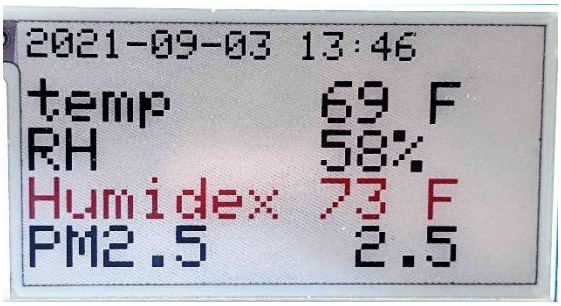


Figure 1, annunciator screen

The annunciator circuitry is trivial, and I do not describe it further.

1.4 Carport Station[8]

The hardware and software of the Carport Station are somewhat similar to those of a peripheral station, but the Carport Station has special real-time requirements. In this document, most statements about peripheral stations apply equally to the Carport Station.

1.5 Base Station[9]

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 battery-backed DS1307 calendar clock; and
- an XBee radio transceiver.

The Base Station is meant to run continuously, with occasional connections to a special-purpose program running on the desktop computer.

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.

1.6 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 6 for details.
[9] See Section 8 for details.
[10] See Section 9 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 that is high when voltage is absent, low when 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 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 cameras

VC0706 motion-detector/cameras[16] 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.

---

[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.itead.cc/wiki/VC0706_UART_Camera_%EF%BC%88Supports_JPEG%EF%BC%89 (accessed 2021-05-25).

2.6 CO_2 level

An SCD30[17] or SCD40[18] sensor can measure temperature, relative humidity, and CO_2 levels. Each of these sensors relies on a standard I^2C interface.

2.7 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 positive and not more than 3.3 V.

2.8 depth

To measure the depth of a small pond in my back yard, I use a sensor from Milone Technologies.[19] The Milone sensor is a clever pressure-to-resistance transducer, so all the peripheral board needs to do is to provide a voltage divider.

2.9 light

The photocell[20] I use outside varies its resistance from a few hundred ohms up to megohms. A photocell tends to last only a year or so in this application.

2.10 rainfall

Every time its bucket tips, my rain gauge[21] 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[22] with an oscillator frequency of 100 Hz or so eliminates these false counts.

2.11 temperature, relative humidity, and dewpoint

I use DHT22[23] temperature-humidity sensors to measure temperature and relative humidity; my software computes a dew point whenever a temperature and a relative humidity are available.

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,[24] 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.

---

[17] 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).

[18] See https://www.sensirion.com/fileadmin/user_upload/customers/sensirion/Dokumente/9.5_CO2/Sensirion_CO2_Sensors_SCD4x_Datasheet.pdf and https://www.adafruit.com/product/5187 (both accessed 2021-09-17).

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

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

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

[22] See Section 4.1 below.

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

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

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.

2.12 wind direction

My wind vane[25] presents its reading as a resistance between 891Ω and 120K.

2.13 wind speed

My anemometer[26] 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[27] with an oscillator frequency of 1 kHz or so eliminates these false counts.

3 effectors

3.1 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 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 uses

- the common ground and the switched 5V supply,
- the common ground and the switched 12V supply, or
- 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, a heater in the shop, and the electroluminescent wire in the carport.

3.2 string of RGB LEDs

A string of RGB LEDs is connected to the Carport Station. The color of the string fades from one color to another in a season-appropriate manner (red/green in December, red/white on Canada Day, and so

---

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

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

[27] See Section 4.1 below.

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

on). The necessary timing of the color fade is not consistent with the structure of the peripheral-station program, so the Carport Station has its own code.

3.3 electroluminescent wire

A string of electroluminescent wire is also controlled by the Carport Station, illuminated as needed, but sparingly to preserve the service life of the wire.

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.)

4.1 debouncing

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 generates an oscillator signal, and then it passes through only signals longer than 4 times the oscillator period. A capacitor external to the MC14490 determines the frequency of the oscillator, which must be set to optimize selection of the anticipated signals.

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.

[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).

5 peripheral stations

5.1 hardware

5.1.1 PCB

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

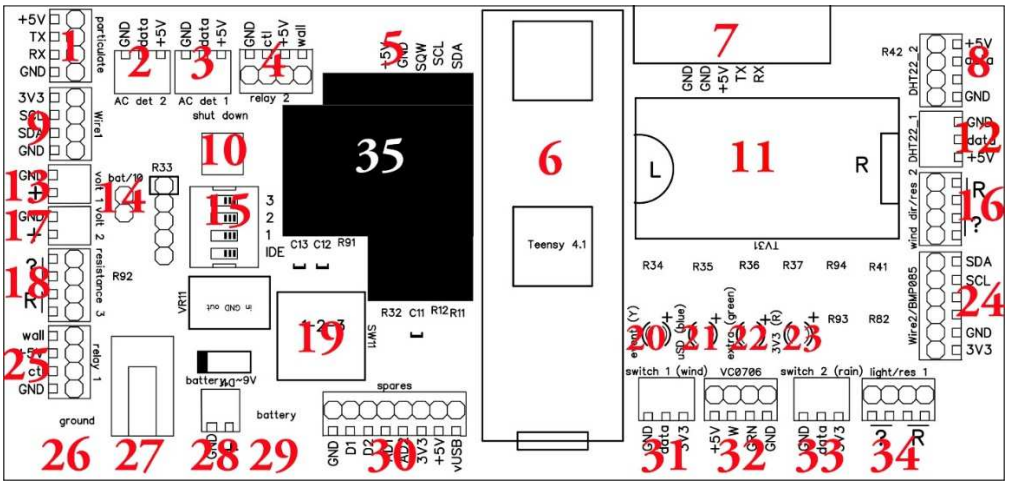


Figure 2 peripheral-station PCB

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

5.1.1.1 *Core circuitry*

The Teensy CPU is here at **6**. 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 **30** ("spares") to restore the connection. The µSD card of the Teensy is formatted as a FAT32 volume.

A 4-line TeensyView display unit is at **11**.

The DS1307 clock is connected at **5;** using controls on its Dashboard,[34] 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 blank shape at **35**, 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 **7.** The XBees in my system are old XBP24s and a few newer XBee3s.[35] I use Digi's XCTU utility[36] 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

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

[34] See Section 9.2.1 below.

[35] 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.

[36] See https://www.digi.com/products/embedded-systems/digi-xbee/digi-xbee-tools/xctu (accessed 2021-05-25).

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 **15** 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 **20** flashes whenever one of the attached sensors has provided data. The blue LED at **21** is illuminated when there are data waiting in the µSD memory. The green LED at **22** is used for debugging. The red LED at **23** is illuminated whenever 3V3 power is available.

Pushing the button at **10** signals that the station should be shut down

5.1.1.2 *power-related*

The three-position switch at **19** selects the board's power source, from among the barrel connector (**27**), a battery (**28**), and the Teensy's USB connector. There are test points for ground at **26** and for the battery voltage (after a diode drop) at **29**.

The connector at **30** provides access to ground, +3V3, and +5V. Its rightmost pin is used for a wire that has been connected to the Teensy's (off-row) **VUSB** pin.

5.1.2 connectors for off-board components

5.1.2.1 *UARTs*

The connectors at **1** ("particulate") and **32** ("VC0706") 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[37] and a VC0706 camera.[38] Accordingly, the pins of these connectors are labeled from the points of view of these attached devices:

- The **UART3 RX** and **TX** pins at **1** are labeled **TX** and **RX** *[sic]*, because they will be connected to the SPS30's **TX** and **RX** wires, respectively.
- The **UART2 RX** and **TX** pins at **32** are labeled **W** and **GRN**, because they will be connected to the VC0706's white and green wires, respectively.

5.1.2.2 *digital input*

The connectors at **2** ("AC det 2"), **3** ("AC det 1"), **8** ("DHT22_ 2"), **12** ("DHT22_ 1"), **31** ("switch 1 (wind)"), and **33** ("switch 2 (rain)") receive a single digital input each. They could be used for pushbuttons, latched switches, or any other generators of binary data, but I now reserve them as follows:

- The connectors at **2** and **3** are reserved for use with detectors of AC voltage.[39]
- The connectors at **8** and **12** are reserved for use with DHT22s.[40] The unconnected pin at **8** is a spacer, allowing a DHT22 to be connected directly to the board without cabling.

[37] See Section 2.2 above.
[38] See Section 2.52.2 above.
[39] See Section 2.12.2 above.
[40] See Section 2.112.2 above.

- The connectors at **31** and **33** are reserved for my anemometer[41] and my rain gauge,[42] respectively. As noted with the descriptions of those sensors, these connections pass through debouncers before getting to the peripheral PCB.

In addition, the pins labelled **`AD1,`** **`AD2`**, **`D1,`** and **`D2`** at **30** ("spares") could be used for other digital input.

5.1.2.3 *digital output*

The connectors at **4** ("relay 2") and **25** ("relay 1") provide access to digital output from the Teensy. The power pins on these connectors facilitate their use with the relays described in Section 3 above.

5.1.2.4 *I²C devices*

The connectors at **9** ("Wire1") and **24** ("Wire 2/BMP085") provide access to the Teensy's **`Wire1`** and **`Wire2`** I²C channels, respectively. These connectors are reserved for a CO2 sensor[43] and a BMP085,[44] respectively; the spacing at **24** allows a BMP085 to be connected to the PCB without cabling.

5.1.2.5 *analog input*

The connectors at **13** ("volt 1") and **17** ("volt 2") allow analog input to the Teensy. The pins labelled **`AD1`** and **`AD2`** at **30** could be used for other analog input.

If the jumper at **14** is installed, then the voltage read at **13** will be a fraction (nominally one tenth) of the voltage (less one diode drop) of the battery connected at **28**.

5.1.2.6 *resistance*

Each of the connectors at **16** ("wind dir/res 2"), **18** ("resistance 3"), and **34** ("light/res 1") allows an unknown resistance (**`?`**) to be connected as the lower limb of a voltage divider, with the upper limb connected as **`R`**. These three connectors are now used for my wind vane,[45] depth gauge,[46] and photocells,[47] respectively, but they could be used for any analogous sensors.

5.2 Software

The software of the peripheral stations is about 9000 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 2.132.2 above.
[42] See Section 2.102.2 above.
[43] See Section 2.62.2 above.
[44] See Section 2.42.2 above.
[45] See Section 2.12 above.
[46] See Section 2.8 above.
[47] See Section 2.9 above.

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). Many of the values needed are times, expressed in milliseconds or microseconds, so suffixes are recognized to make large numbers less unwieldy. For example, the relay connected to one of my peripheral stations is configured with

```
// ----waterfall

relay_1.usNominalReadInterval   =   15sK
relay_1.msNominalReportInterval =    5m
relay_1.FollowSwitch            =    0
relay_1.Cycler                  =    1
relay_1.Thermostat              =    0
relay_1.msMinimumHold           =    1m
relay_1.OnTime                  = 0800
relay_1.OffTime                 = 2200
relay_1.Singleton               =    0
```

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. The pertinent section of one of my **X_config.txt** files is

```
// ********* sensors ******************

ACDetect_1.InUse       =  0
ACDetect_2.InUse       =  0
BMP085.InUse           =  1
DHT22_1.InUse          =  1      // pond
DHT22_2.InUse          =  1      // outside temp & RH
relay_1.InUse          =  1      // waterfall
resistance_1.InUse     =  1      // photocell
resistance_2.InUse     =  0
resistance_3.InUse     =  0
SPS30.InUse            =  0
switch_1.InUse         =  0
switch_2.InUse         =  0
VC0706.InUse           =  0
voltage_1.InUse        =  1      // current transformer L
voltage_2.InUse        =  1      // current transformer R
```

5.2.2 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.[48]

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. 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.[49]

5.2.3 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

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

[49] For more on the Desktop-to-Base messages, see Sections 8.2.1 and 9.58.2.1 below.

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.

5.2.3.1 *board slots*

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

```
//  0          1              2            3          4
bsBogus, bsACDetect_1, bsACDetect_2, bsBMP085, bsDepth,
bsDHT22_1, bsDHT22_2, bsLightningNOTUSED, bsRelay_1,
bsResistance_1, bsResistance_2, bsResistance_3NOTUSED, bsSPS30,
bsSwitch_1, bsSwitch_2, bsVC0706, bsVoltage_1, bsVoltage_2,
bsRelay_2, bsRelay_3, bsSwitch_3, bsSwitch_4, bsWire_1,
bsVoltage_3
```

In other words, it has one entry for each of the potential sensor (or relay) connections to the peripheral board. If different or expanded peripheral boards were added to the system, additional entries would be made in this file. Some codes are not used, but they are preserved for backwards compatibility.

5.2.3.2 *WhatMeasured*

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

```
//    0          1                2              3        4
wm2Bogus, wm2Undefined, wm2PressureInches, wm2RH, wm2TempF,
wm2DepthInches,  wm2LightningDistanceNOTUSED,
wm2LightningDistPerHourNOTUSED, wm2LightningEnergyNOTUSED,
wm2LightningFlashesNOTUSED, wm2LightningNoisePerHourNOTUSED,
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
```

These entries are used to tag recorded data. They mostly correspond many-to-one to specific sensor types, but the correspondence is imperfect. and some codes are used by the system for administrative purposes not directly related to sensor data. Some codes (*e.g.,* all of the lightning-related one) remain on the list only for backward compatibility.

5.2.3.3 *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[50] of 65.42°F from a DHT22 or BMP085 might appear as the triple

4\t6542\t2

5.2.3.4 *sensor tags*

For local purposes of the peripheral station, each sensor is assigned a single-character tag, mnemonic if possible. The **X_config.txt** files of all my peripheral stations include the lines

```
ACDetect_2.SensorTag    =  'A'
BMP085.SensorTag        =  'B'
SCD40.SensorTag         =  'C'
DHT22_2.SensorTag       =  'D'
DHT22_1.SensorTag       =  'E'
ACDetect_1.SensorTag    =  'F'
resistance_1.SensorTag  =  'L'
voltage_1.SensorTag     =  'P'
voltage_2.SensorTag     =  'p'
switch_2.SensorTag      =  'R'
SPS30.SensorTag         =  'S'
relay_1.SensorTag       =  'T'
switch_1.SensorTag      =  'W'
resistance_3.SensorTag  =  'w'
```

but different stations could use different sets of tags.

5.2.3.5 *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'
```

5.2.3.6 *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

<YYYY-MM-DD HH:MM:SS>**\t**<sensor tag>**\t**<triple>

so that the temperature reading described in Section 5.2.3.3 above might (if it had been made by the DHT22 connected at 8) appear as

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

5.2.3.7 *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

[50] 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.

the Base Station only after a sensor 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

5.2.3.8 *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

```
resistance_1.Damping                  =  0.9
resistance_1.usNominalReadInterval    =  1mK
resistance_1.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 paragraphs 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.

5.2.3.9 *singleton*

If the "Singleton" property is true of a sensor, then the sensor driver may cause a single timestamped, formatted value to be shown on the station's TeensyView display whenever this sensor reports. In the case of a sensor that provides multiple data values at each reading, the value displayed is chosen by the sensor driver. For example, the

```
DHT22_2.Singleton = 1
```

line in one of my **X_config.txt** files causes that station's TeensyView to become a time-and-temperature display. The DHT22 driver could have been written so as to facilitate time-and-humidity displays, but it wasn't.

5.2.3.10 *the **tSensor**[51] class*

As part of its initialization, the peripheral-station program constructs 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

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

activity of the central loop of the peripheral-station program is to interrogate each active driver, to see if that driver is due to interact with its associated sensor.

Each sensor driver is an instance of a class specific to one type of sensor. Thus, there is a **tACDetector** class, a **tBMP085** 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. Most of most **setup** methods consists of note-taking that may later be useful for debugging.

Every active sensor's **loop** procedure is called on every circuit of the peripheral station's central loop. If a shutdown is in progress,[52] then the sensor's **ShutSensorDown** method is called; **tSensor**'s own **ShutSensorDown** method is null. Otherwise, if the sensor is due to be read, then **loop** calls the sensor's **ReadTheSensor** method.

5.2.4 sensor drivers

5.2.4.1 *AC Detection*

The MID 400 chips used for AC detection send 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. Reports from this driver use the **wm2IsNowOn** tag.

5.2.4.2 *BMP085*

Most of the BMP085 driver is code copied from a Teensy forum.[53] Reports from this driver use the **wm2TempF** and **wm2PressureInches** tags. The temperature reported by a BMP085 may be used to control a local or remote relay,[54] and the reported pressure may be used by a local or remote CO_2 sensor.[55]

5.2.4.3 *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. The temperature reported by a DHT22 may be used to control a local or remote relay.[56] Reports from this driver use the **wm2TempF** and **wm2RH** tags.

---

[52] See Section 5.1.1.1 above.
[53] See https://forum.pjrc.com/threads/17143-I2C-barometer-BMP085?highlight=bmp085 (accessed 2021-05-25).
[54] See Section 5.2.5.3 below.
[55] See Section 2.6 above.
[56] See Section 5.2.5.3 below.

5.2.4.4 *resistance*

This sensor driver estimates an external resistance by using it as the lower arm of a voltage divider. The upper arm of the divider is a known external resistor whose value is recorded in an **UpperOhms** line in **X_config.txt.** Accumulated results are exponentially damped before being reported with a **wm2Resistance** tag.

On top of that simple pattern, two special cases are recognized. If the **Light** parameter is set to 1 in **X_config.txt,** then the resistance being measured is assumed to be that of a photocell. The "resistance" reported will then be transformed into a value in [0, 100], using the **X_config.txt** parameters **Light000** and **Light100**. A light level may be used in control of a relay.

If the **WindDirection** parameter is true, then the "resistance" reported will be a value in [0, 360). To compute this value, the driver makes use of specifications provided in the wind vane's data sheet,[57] which describes the (highly nonmonotonic) relation between the vane's direction and the resistance of the connection.

To perform the damping that combines several readings of wind direction before reporting, the scalar direction is converted to a ($W\cos\theta$, $W\sin\theta$) vector, where W is the most recent measured wind speed (or 1, if no recent wind speed is available). . The vector is converted back to a directional scalar before the report is made.

5.2.4.5 *SCD30, SCD40*

These drivers try to provide their sensors' firmware with the current air pressure, but they use a default value (29.72 in Hg) if no current value is available. The temperature reported by one of these sensors may be used to control a local or remote relay.[58]

The SCD30/SCD40 data are reported with the **wm2CO2ppm, wm2RH**, and **wm2TempF** tags.

5.2.4.6 *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.

5.2.4.7 *switch closure*

Switch closure sounds simple, but this driver is in fact the most complex of the current sensor drivers. This is because the behavior of a switch can be reported

- as open or closed whenever reporting is due, or
- whenever the state has changed, or
- by counting closures (for example, bucket tips of a rain gauge), or
- by observing the closure frequency (for example, the frequency of a spinning-cup anemometer).

[57] See https://www.argentdata.com/files/80422_datasheet.pdf (accessed 2021-05-25).
[58] See Section 5.2.5.3 below.

5.2.4.7.1 *state monitor (**LogState = 1**)*

When the **LogState** parameter is true, then the station uses the **wm2SwitchClosed** tag to report the switch being closed (1) or open (0). The reporting interval should in this case be identical to the reading interval.[59]

5.2.4.7.2 *change logger (**LogChange = 1**)*

When the **LogChange** parameter is true, then the station uses the **wm2SwitchClosed** tag. It reports at the specified reporting interval, but it also reports whenever the switch is read if the switch's state has changed since the last report.

5.2.4.7.3 *event counter (**EventCounter** = 1)*

When the **EventCounter** parameter is true, then the driver takes account of the **X_config.txt** parameters **JammedLowDefined, usJammedLowThreshhold, ReportNewEvents, ReportCumulativeEvents, ReportEveryNEvents,** and **ReportEventsByTime.**

If

- the **JammedLowDefined** parameter is true,
- the switch has remains closed for more than **usJammedLowThreshhold** microseconds, and
- the apparent jam has not yet been reported

then the driver reports the value of **wm2Jammed** to be true.

If **ReportCumulativeEvents** is true, then the total event count will be reported, using the **wm2EventCount** tag. If **ReportNewEvents** is true, then the count of events since the last report will be reported, using the **wm2NewEvents** tag.

When the **ReportEventsByTime** parameter is true, then a report is made whenever the report interval has elapsed. When the value of the **ReportEveryNEvents** parameter is non-zero, then a report is made whenever that many events have occurred since the last report.

5.2.4.7.4 *frequency counter (**FrequencyCounter** = 1)*

When the **FrequencyCounter** parameter is true, then the driver takes account of the **X_config.txt** parameters **MaxPlausibleHz** and **WindSpeed.** If **WindSpeed** is true, then the driver looks also at **WindMPHPerHz**, and all of the "frequencies" reported are actually the products of the measured frequency and the value of **WindMPHPerHz**.

When it's time to report, the driver reports

- the exponentially-damped accumulated readings, using the **wm2Frequency** tag;
- the lowest frequency reported in any reading since the last report, using the **wm2FrequencyMin** tag;
- the highest frequency reported in any reading since the last report, using the **wm2FrequencyMax** tag; and
- if **WindSpeed** is true, the most recent frequency read, using the **wm2FrequencyRaw** tag.

[59] I might change this to behave as at present when the reading and reporting intervals are identical, but to report the duty cycle (that is, fraction of time closed) when the reporting interval is longer.

5.2.4.8 *voltage*

The voltage-sensor driver uses only the **wm2Voltage** tag to report its result, but it may, depending on what it finds in **X_config.txt,** transform the incoming data before reporting.

If **BatteryTest** is true, then the driver assumes that the shunt at 14 is in place, and that the voltage delivered at 13 is a divided version of the voltage (after a diode drop) of the battery at 28. Components on the peripheral-station board are chosen so that the divider has a ratio of about 0.1, but the actual ratio should be measured and then supplied in a line like

```
Board.BatteryDivider = 0.0824
```

In order to allow decentralized compensation for external, component-dependent voltage adjustment, the voltage finally reported by the driver is (estimated voltage) × **MultiplyBy + ThenAdd.**

5.2.4.9 *VC0706 camera*

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 9.6 below, it's also possible to direct the taking of a VC0706 image from the Desktop Program.

5.2.5 relay control

The relay on a peripheral board can be controlled directly by commands from the Desktop Program. Also, the peripheral station can control the relay autonomously, as specified in the **X_config.txt** file.

When the relay is configured by **X_config.txt,** exactly one of **FollowSwitch, Cycler, Thermostat, and Worklight** should be true. The **msMinimumHold** parameter specifies the minimum time between state changes.

5.2.5.1 *following a switch*

If **FollowSwitch** is true, then the relay is activated whenever the switch specified by **SwitchToFollow** is closed (that is, whenever an indicated Teensy pin is grounded). The recognized values of **SwitchToFollow** are

SwitchToFollow	Teensy pin	location	identifier
1	3	31	switch_1
2	9	33	switch_2
3	36	30	spares/D1
4	37	30	spares/D2

5.2.5.2 *daily cycle*

If **Cycler** is true, then lines like

```
relay_1.OnTime  = 0700
relay_1.OffTime = 2100
```

specify the on-time and off-time for daily activation of the relay. An off-time nominally earlier than the on-time will be correctly interpreted to arrange operation across midnight.

I am using a cycler-mode relay to control the pump that drives a waterfall in a backyard pond. To keep the pump from being asked to move ice, the relay-supporting code recognizes

```
relay_1.WhichRecentTemp = 2 // 1, 2 DHT22s; 3 BMP085
relay_1.TooColdPossible =    1
relay_1.WarmEnoughF     =   40
```

and refrains from running the pump when the air temperature is below 40°F.

5.2.5.3 *thermostat*

If **Thermostat** is true, then the relay will be configured as a thermostat, controlling a heating device. The thermostat can try to achieve a certain target temperature all day, or — if the **OnTime** and **OffTime** parameters are supplied —only during certain hours. In the latter case, the thermostat will attempt to maintain a "fallback" temperature during the off hours. To avoid high-frequency cycling around the target temperature, the thermostat will observe a hysteresis rule: It will keep the heater on until the temperature is a certain amount above the target temperature, then not turn it on again until the temperature is the same amount below the target temperature.

The target temperature, fallback temperature, and hysteresis amount are set with the parameters **TargetF, FallbackF**, and **HysteresisF**, respectively.

The temperature followed by the relay/thermostat is determined by the **WhichRecentTemp** parameter. The recognized values of **WhichRecentTemp** are

WhichRecentTemp	temperature source
1	DHT22_1 (at 12)
2	DHT22_2 (at 8)
3	BMP085
4	SCD40
5	SCD30

5.2.5.4 *worklight*

If **Worklight** is true, then the relay will be configured to control a sort of nightlight. That is, it will be turned on between certain hours when the ambient light level is low. For example, the pertinent lines in my living-room's **X_config.txt** file are

```
relay_1.Worklight                =   1
relay_1.usStandardReadInterval   =  15KK
relay_1.msStandardReportInterval =   2m
relay_1.msMinimumHold            =   5m
relay_1.OnTime                   = 0800
relay_1.OffTime                  = 2359
relay_1.LightEnough              =   36
relay_1.LightHysteresis          =    3
```

6 Carport Station

The Carport Station's PCB carries three STP16FN06 MOSFETs to drive the string of RGB LEDs, and a small relay to control the electroluminescent wire. If, in the current spirit of the system, these components were deported to satellite boards, then the Carport Station could use a standard peripheral board, but it would still need to have its own program.

The software of the Carport Station (about 1400 lines of C++) is similar to that of the peripheral stations, but not at the top level. At that level, the Carport Station spends most of its time managing the string of RGB LEDs, breaking every once in a while

- to make temperature measurements with a DHT22 and
- to control the relay that provides power to the electroluminescent wire. The EL wire is powered when and only when
  - the time is within the **EL.OnTime** and **EL.OffTime** bracket set in its **X_config.txt**,
  - the light level of its photocell has been below **EL.TooMuchLight** while the time was in that bracket today, and
  - an enabling message has been sent from the Desktop Program.

The idea here is to enable the EL strip only as needed, and not to have transient carport lights turn the EL wire off after it has been properly turned on for the evening.

7 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.

8 Base Station

8.1 Hardware

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

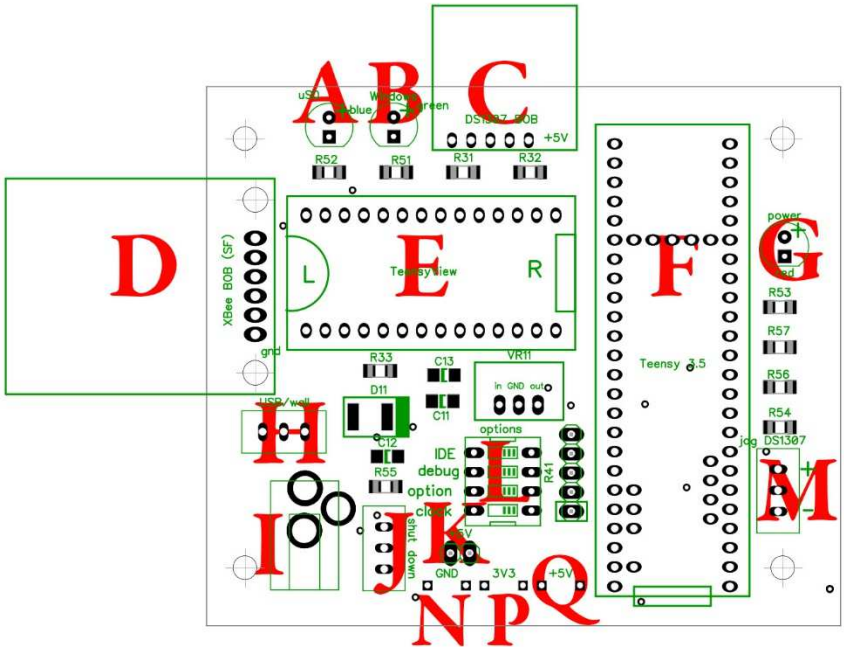


Figure 3 Base-Station PCB

The schematic and the DipTrace schematic and layout files are available on my Web site.[60]

Many of the components here (XBee at **D**, TeensyView at **E**, Teensy 3.5 at **F**, barrel connector at **I**, ground testpoint at **N**) are the same as those used in the peripheral stations. Testpoints for the +3V3 and +5V rails are provided at **P** and **Q**, respectively, and an ammeter shunt for the +5V supply is provided at **K**.

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.

The switch at **M** is no longer used.

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

8.2 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.

8.2.1 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[61]

```
   wtbEMPTYNOTUSED, wtbConnect, wtbDeleteJPEGs, wtbDisconnect,
wtbInitializeuSD, wtbInterrupt, wtbPeripheralRebootNOTUSED,
wtbReadTime, wtbRelayCycle, wtbRelayFollowSwitch, wtbRelayOff,
wtbRelayOn, wtbRelayRunDaily, wtbRelayRunOnceNOTUSED,
wtbSendToXBee, wtbSetTime, wtbShutDown, wtbStartDump,
wtbTakeVC0706Image, wtbTargetF, wtbThermostat,
wtbThermostatRelaxNOTUSED, wtbTickleBaseStation, wtbAnnunciator,
wtbBroadcastPressure, wtbBroadcastOutsideTempF,
wtbBroadcastLocalTime, wtbSpeedupChanged, wtbEnableELWire
```

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

```
   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
```

8.2.1.1 *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.

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.

[61] Some items are obsolete, retained only for backward compatibility.

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.**

8.2.1.2 *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, btwXBeeOK, btwStartingFileSystem, btwSetupComplete

or reporting difficulty

btwXBeeBufferBusy, btwXBeeSerialNumberFailed, btwXBeeDifficultCommandMode, btwXBeeMYFailed, btwXBeeNetIDFailed, btwXBeeNIFailed, btwXBeeStuckInCommandMode, btwXBeeNoCommandMode, btwFileSystemTrouble, btwXBeeNoSetDestination.

8.2.1.3 *directives to the Base Station*

The **wtbInitializeuSD** message directs the Base Station to delete all files on its µSD card.

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**. A **wtbShutDown** message directs the Base Station to flush its buffers onto the µSD card; this message was intended to allow the Base Station to be shut down with no loss of data, but this scenario is not yet supported.

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.

8.2.1.4 *messages passed to specific peripheral stations*

As described in Section 9.2 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 **wtbBroadcastLocalTime, wtbDeleteJPEGs, wtbRelayCycle, wtbRelayFollowSwitch, wtbRelayOff, wtbRelayOn, wtbRelayRunDaily, wtbRelayRunOnce, wtbTargetF, wtbThermostat, wtbTakeVC0706Image,** and

wtbBroadcastPressure.[62] 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.

8.2.1.5 *messages passed to annunciators*

The **annunciators** module of 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.

8.2.2 data from the peripheral stations

Just before the Base Station begins to transmit data from its µSD card to the Desktop Program, it sends a **btwDumping** message. When it starts passing 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 5.2.3.7 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.[63]

When the Base Station receives a line of data from a peripheral station, it writes the board tag[64] 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,[65] the Base Station sends a **btwDumpEnded** message.

---

[62] The SCD30 and SCD40 CO_2 sensors try 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.

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

[64] See Section 5.2.3.5 above.

[65] See Section 8.2.1.3 above.

9 Desktop Program

The Desktop Program (about 33 000 lines of Delphi) runs in Windows. Its top-level form is small

Figure 4 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**[66] and the panels showing current readings of the various sensors.[67] As described in Section 9.10.2 below, some graphs may also be displayed at this time.

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.

9.1 database

The center of the Desktop Program is an Advantage relational database.[68] Most of the tables of the database are used for data received from the peripheral stations, but some tables are used for administrative purposes.

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 most 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 2024-03-11, for example, the accumulated temperature/pressure/relative humidity measurements were divided among a 2.7MB table of recent data, a 15MB table of older data, and a 292MB table of data from before 2022. The process of moving data out to 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).

[66] See Section 9.2.1 below.
[67] See Section 9.6 below.
[68] See https://dbdb.io/db/advantage-database-server (accessed 2021-05-25).

The data tables currently maintained are

name	content
CO2	CO2 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 target of observation is implicit.

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

9.1.1 non-contributory data

Data for the **Light, OnOff, Rain, Thermostats**, **TRHP,** and **Wind** tables might be acquired frequently, but the acquired values may be mostly unchanged 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 the **OnOff** table only to begin a calendar day or when the data show a change. In the implementation, each *unique* datum in these 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 specific parameters here are obtained from the Windows Registry, where they are tweaked from time to time.

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.

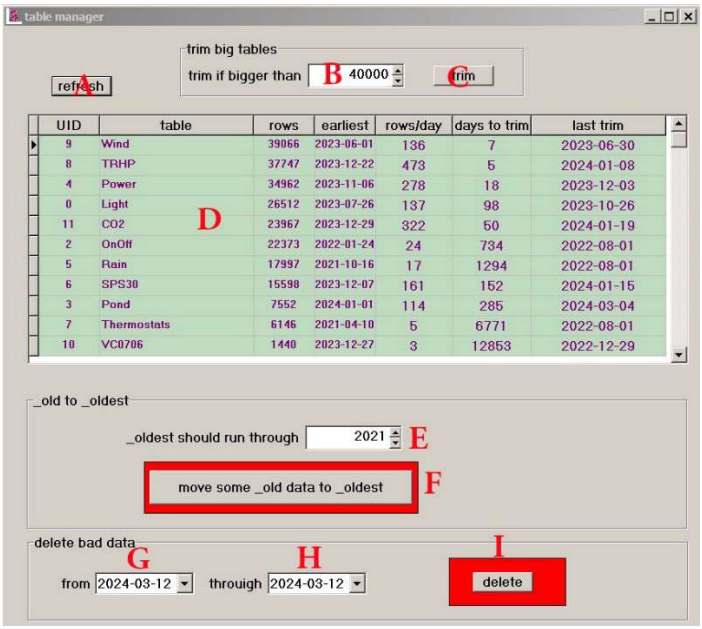


Figure 5, table manager

Here and in other forms, fields displayed with purple letters on a green background (here, all of the fields in the grid) are read-only. 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**.

9.2 input processing

9.2.1 Dashboard

The initial appearance of the **Dashboard** is

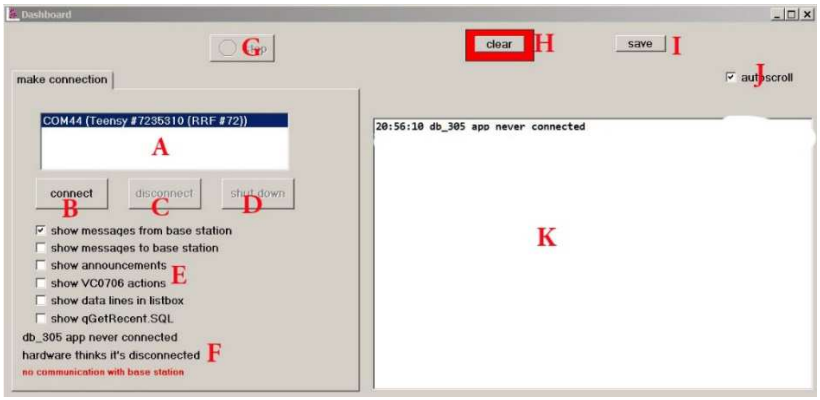


Figure 6 initial appearance of the Dashboard

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:

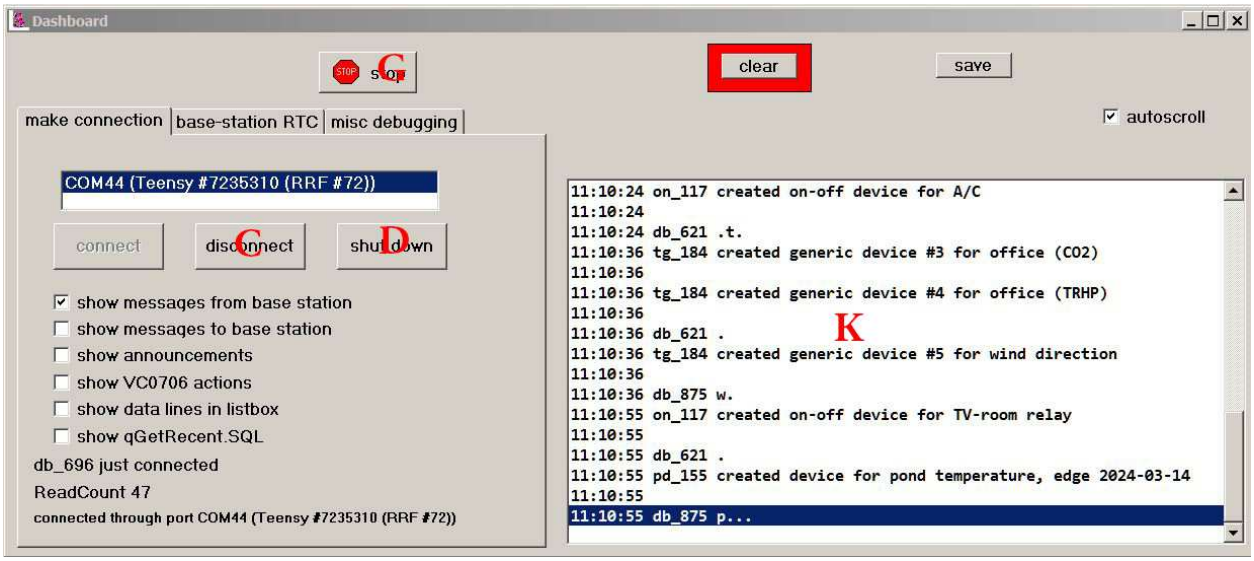


Figure 7 Dashboard after connection to Base Station

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 9.3 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,[69] 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 tools for sending arbitrary messages to the Base Station. These facilities have not been used.

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

9.2.2 Log

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

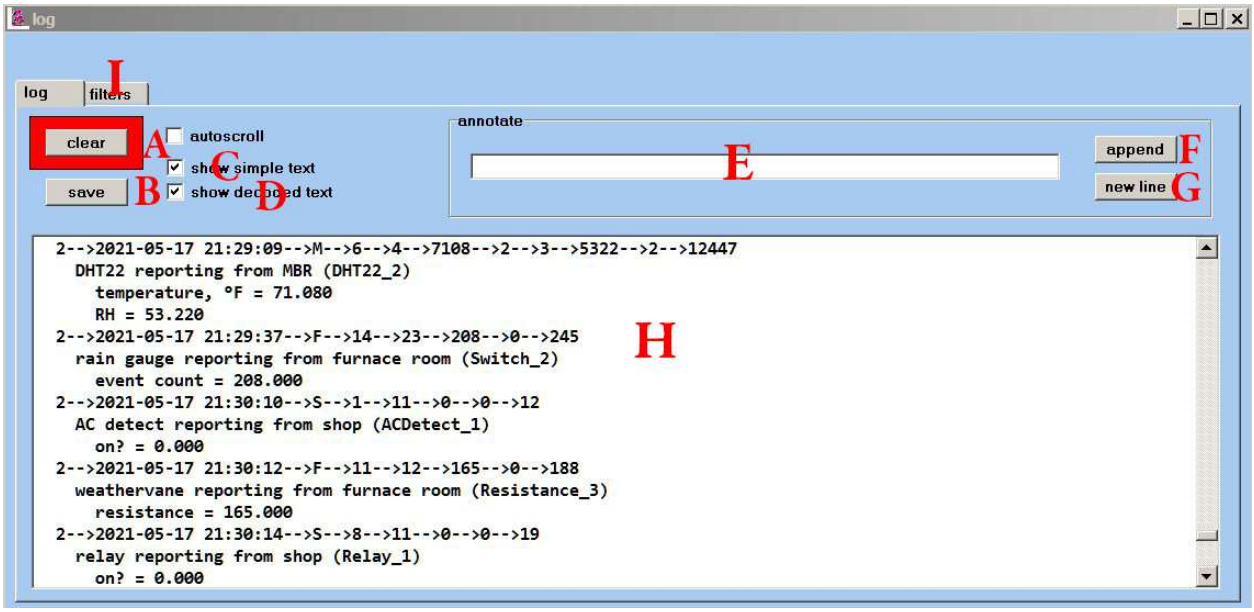


Figure 8 log

Because the boxes at **C** and **D** are here both checked, each received line is reported in two forms. Each of the data lines shown here began with **2**, showing that it had been passed through from its originating peripheral station without pausing on the Base Station's µSD card. Then, after the Base Station's timestamp, there follow the station tag, the board slot tag, the triple(s), and the checksum. The second through fourth lines here at **H** decipher the first: The **M** station tag is that of the master bedroom, slot #6 is the DHT22_2 slot, and 4 and 3 turn out to be **wm2TempF** and **wm2RH,** respectively.

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

[69] See Section 9.1.1 above.

The tab at **I** contains another listbox and controls that allow selectivity (by board, area observed, or sensor type) of the data lines listed.

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

9.3 the Sensor Manager

Clicking **sensors/sensor manager** on the main menu evokes the Sensor Manager form. At **G** on the principal page of this form

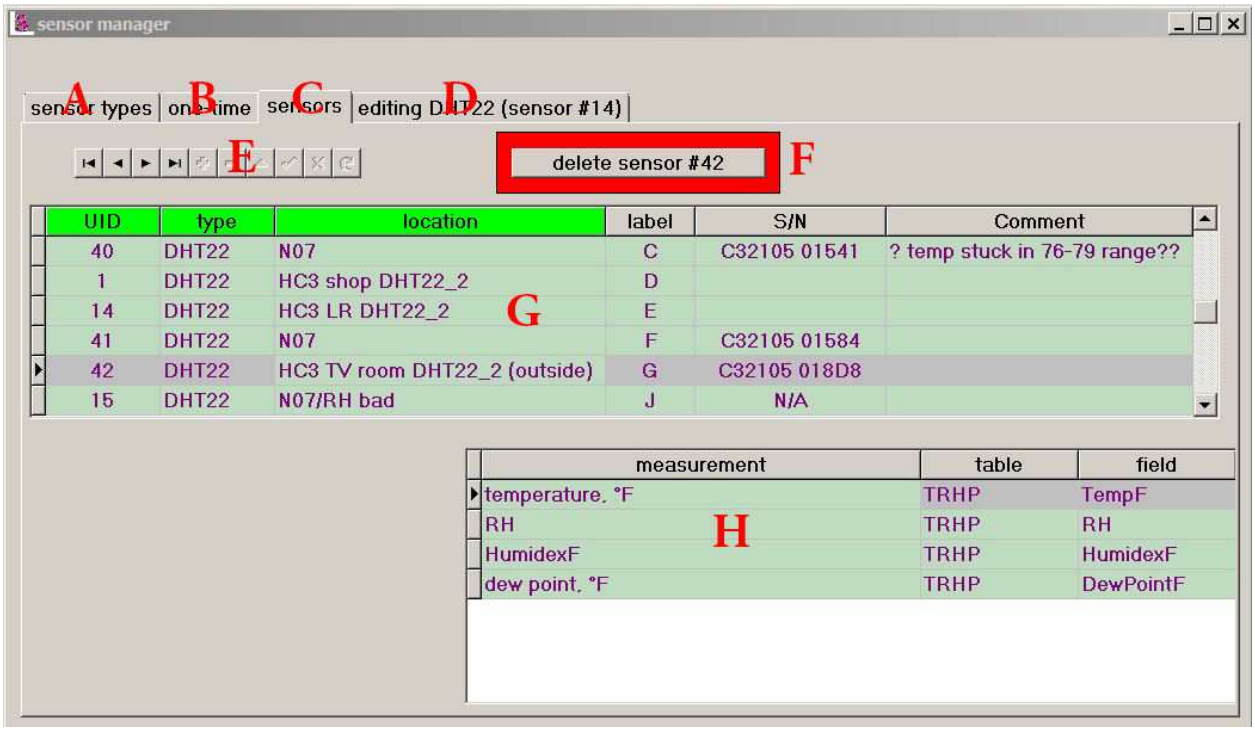


Figure 9, Sensor Manager

there is a list of all of my sensors, whether or not they are in use. The grid at **H** shows how data from the sensor selected at **G** are stored in the database. Double-clicking an entry at **G** allows information about the selected sensor to be edited: where is it, to what slot (if any) of what board (if any) it is connected, how its data is to be stored in the database, and so on.

This figure shows another convention: If a grid's column header has a bright green background, then clicking on that header sorts the records of the grid into ascending order by the entries of that column.

The system allows for the submission of user-generated data that are handled like data coming from sensors. The **sensors/bogus** sensors item on the main menu (below **D** in Figure 4 above) provides this channel. This feature is now used only to log the setting of a little refrigerator whose temperature is followed by the Carport Station's DHT22.

9.4 the Board Manager

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

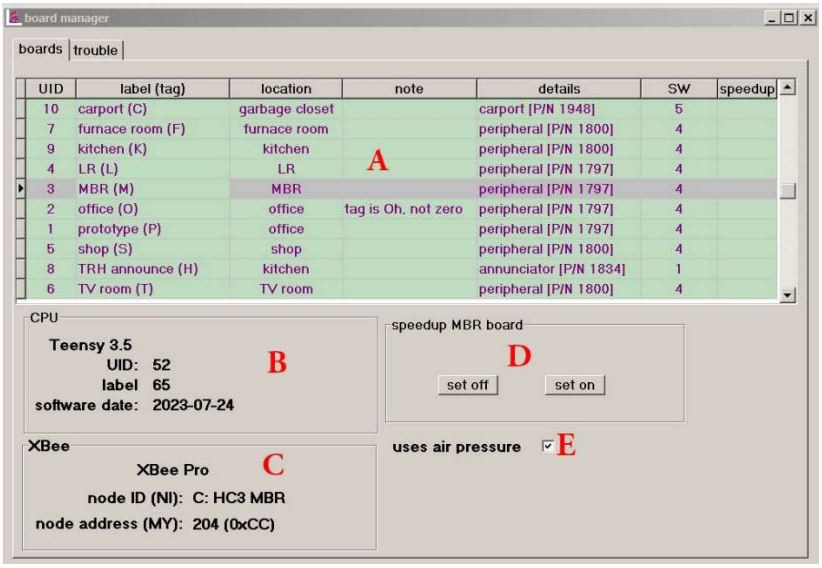


Figure 10, Board Manager

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

As noted in Section 5.2.3.8 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 **D**.

As noted in footnote 62 above, 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 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.[70]

9.5 relay control

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

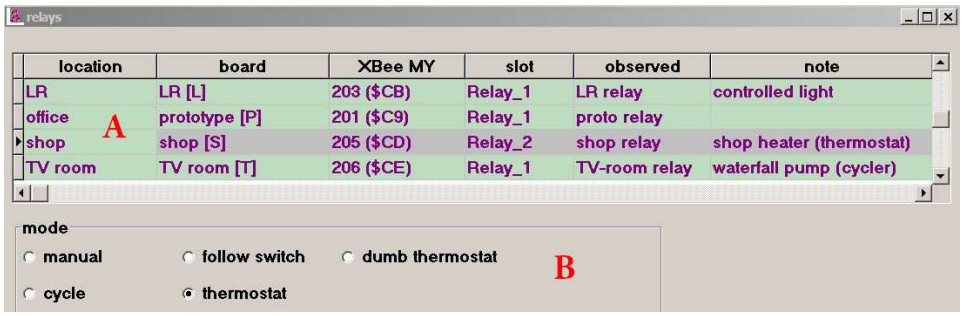


Figure 11, relay form, initial state

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

The peripheral stations that have relays are listed at **A**, and the possible modes of relay operation are options in the radiobox at **B**.

Clicking one of the options at **B** causes the relays form to expand appropriately, giving the user a chance to indicate hours of operation and temperature-related options.

- Selection of the **manual** option causes **on** and **off** buttons to become visible, allowing **wtbRelayOn** and **wtbRelayOff** messages to be sent to the peripheral station selected at **A**.
- If **cycle** is selected, then the user is able to specify that the selected relay should be on for certain hours daily, but only when a specified temperature sensor is reporting temperature above a specified value. This option is used for the pump that creates the recirculating waterfall in the back-yard pond.
- If **follow switch** is selected, then a menu of switches[71] will be presented, and the relay will be set on or off to follow the state of the chosen switch.
- If **dumb thermostat** is selected, then the user is able to specify a temperature sensor and to arrange that (with specified hysteresis of temperature) the relay will be on whenever the temperature is less than a specified value.
- With the **thermostat** option, the user can arrange that the relay will be on for specified hours of the day if a specified temperature sensor sees a temperature below a specified value *TargetF*, and the relay will be on at other times if the sensor sees a temperature below a different (generally lower) value *FallbackF*.

Messages sent to the peripheral station in response to this desktop activity include

wtbTargetF	<*TargetF*>	<10 × hysteresis>
wtbThermostat	<*FallbackF*>	<sensor index>
wtbRelayRunDaily[72]	<on time >	<off time>
wtbRelayCycle	<minimum temp >	<sensor index>
wtbRelayFollowSwitch	<switch index>	
wtbRelayOff		
wtbRelayOn		

9.6 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

[71] See Section 5.1.2.2 above.

[72] Times transmitted in **wtbRelayRunDaily** messages are transmitted as minutes after midnight. Start and stop times that span midnight are correctly handled.

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

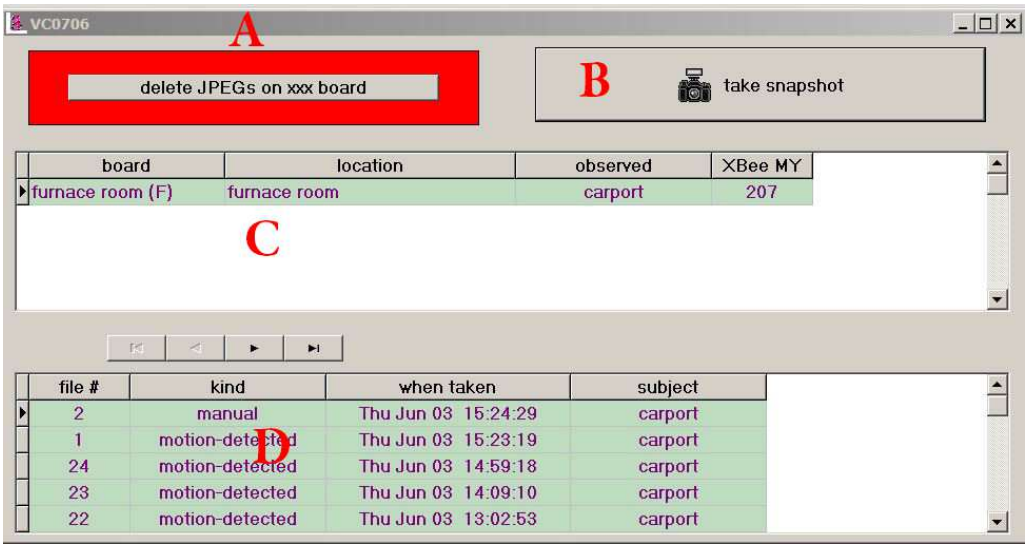


Figure 12, VC0706 image

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.

9.7 sensor displays

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

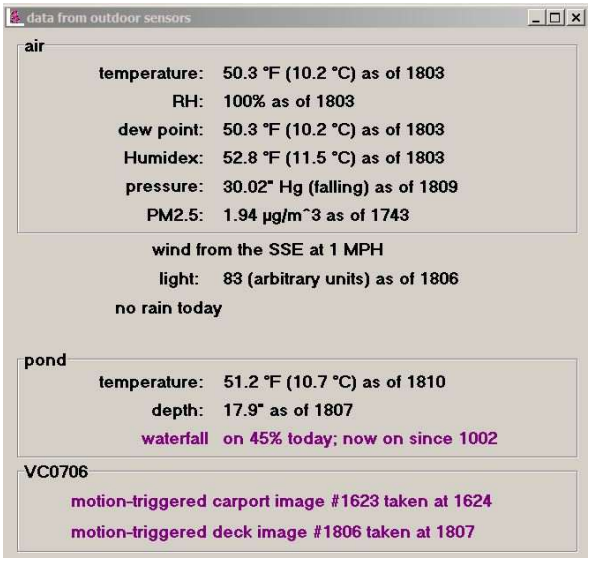


Figure 13, outdoor sensors

and one for the indoor sensors:

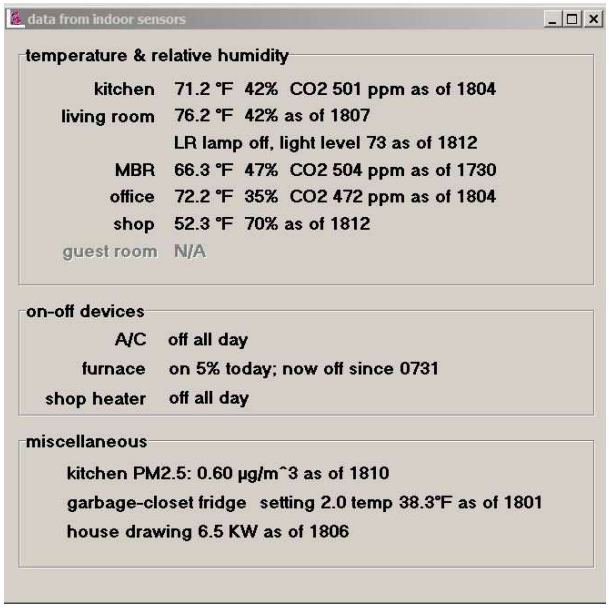


Figure 14, indoor sensors

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`**.

9.7.1 the **`annunciators`** module

The two sensor-display forms provide information to the annunciators module, which in turn organizes the text lines displayed on the annunciators. The **`annunciators`** module contains an enumerated type that identifies data that might be of interest to an annunciator. The enumerated type now is

```
tAnnunciatedVar = (avOutsideHumidexF, avOutsideTempF, avOutsideRH,
                   avPM25);
```

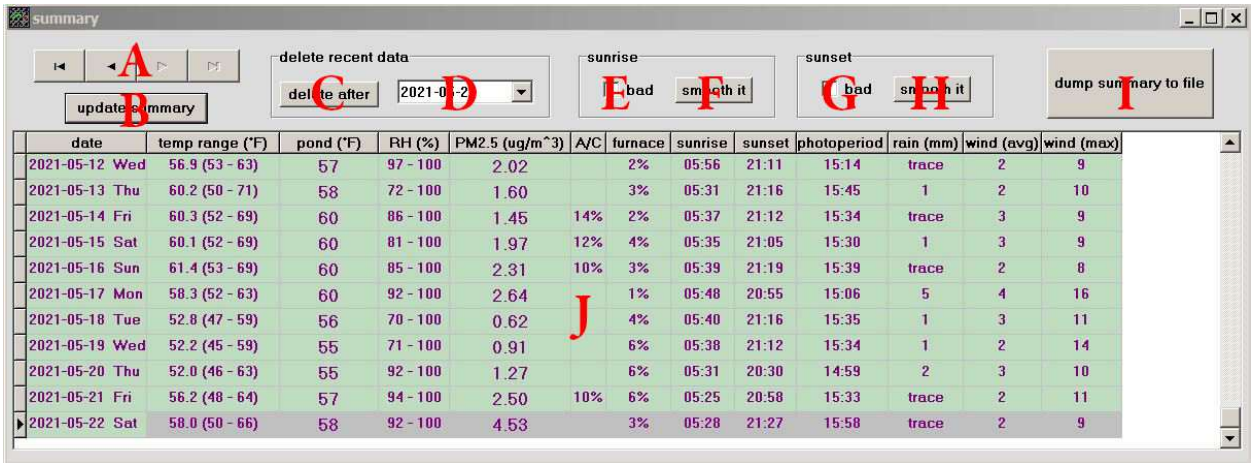
identifying the items shown in Figure 1. The **`annunciators`** module defines a partially-abstract **`tAnnunciator`** class, each instance of which is defined by a board tag[73] and a set of **`tAnnunciatedVar`** items. Each descendant of the **`tAnnunciator`** class is defined to handle a specific display format for a specific set of items.

The sensor-display forms give special treatment to any datum corresponding to one of the **`tAnnunciatedVar`** items. Each such datum is passed to the **`annunciators`** module, where it is stored. If there is a **`tAnnunciator`** instance that has not recently been updated, and whose required data are all available, then that instance uses multiple calls to the **`tAnnunciator.SendLine`** routine to arrange for transmission of formatted lines to the Base Station and eventually to the instance-associated annunciator board.

[73] See Section 5.2.3.5 above.

9.8 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 2024-03-14, my table contains 5161 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 15, 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.

9.9 Extremes

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

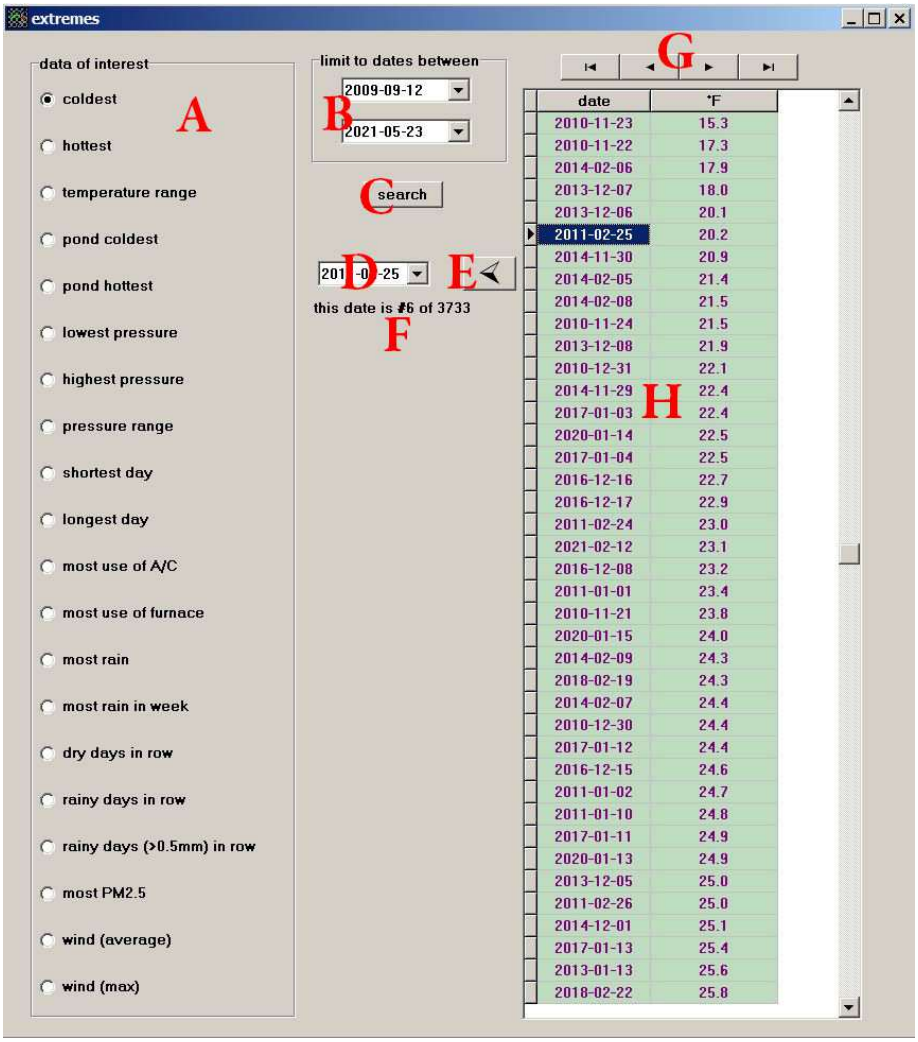


Figure 16, extremes

The button at **C** pulls into the grid at **H** all of the data selected at **A**, taken from within the date range specified at **B** and sorted appropriately. The rank (**F**) of a selected date **D** can be found by clicking on button **E**.

9.10 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 Windows Registry item

`Software\RRF\HomeControl3\DecentGraphInterval`

9.10.1 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.

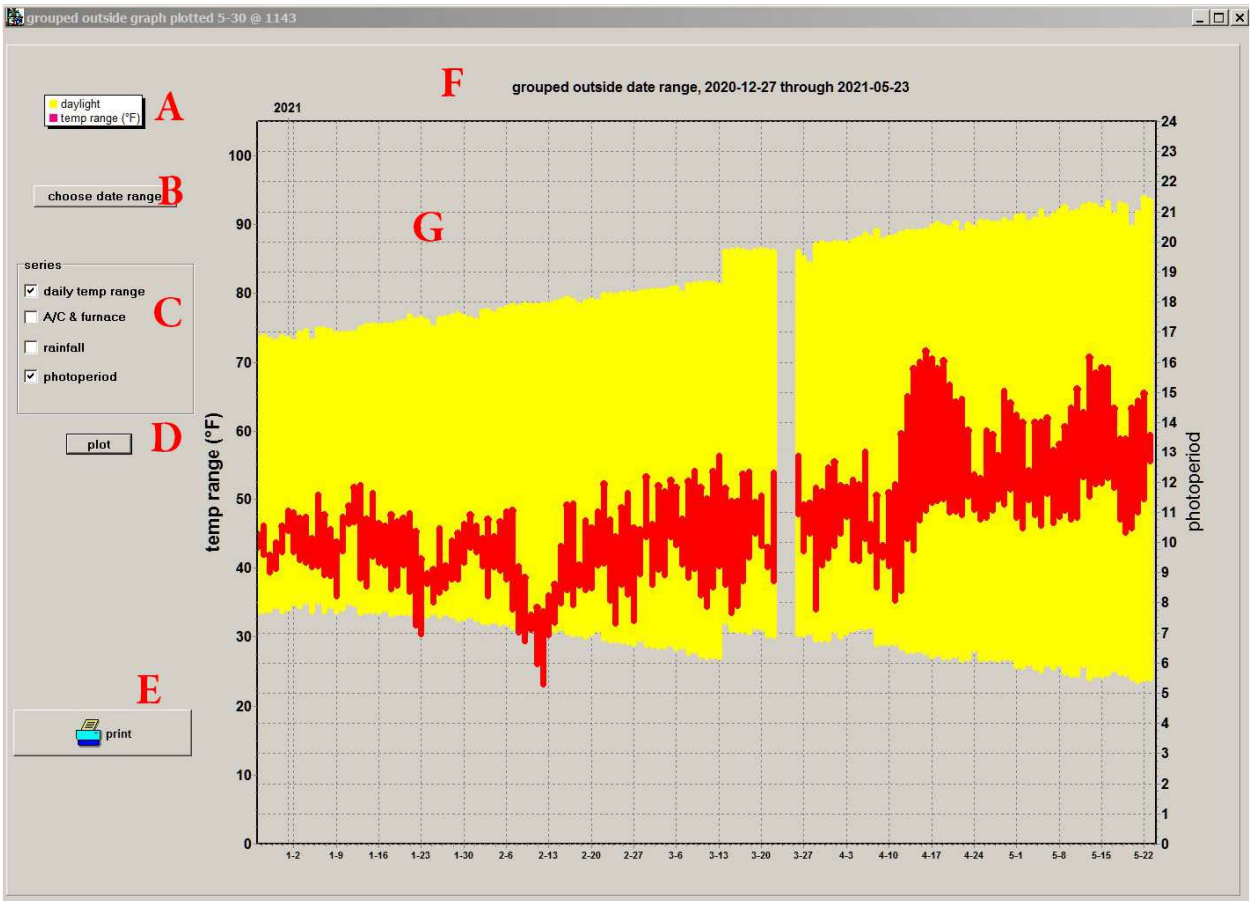


Figure 17graph: grouped outside

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).

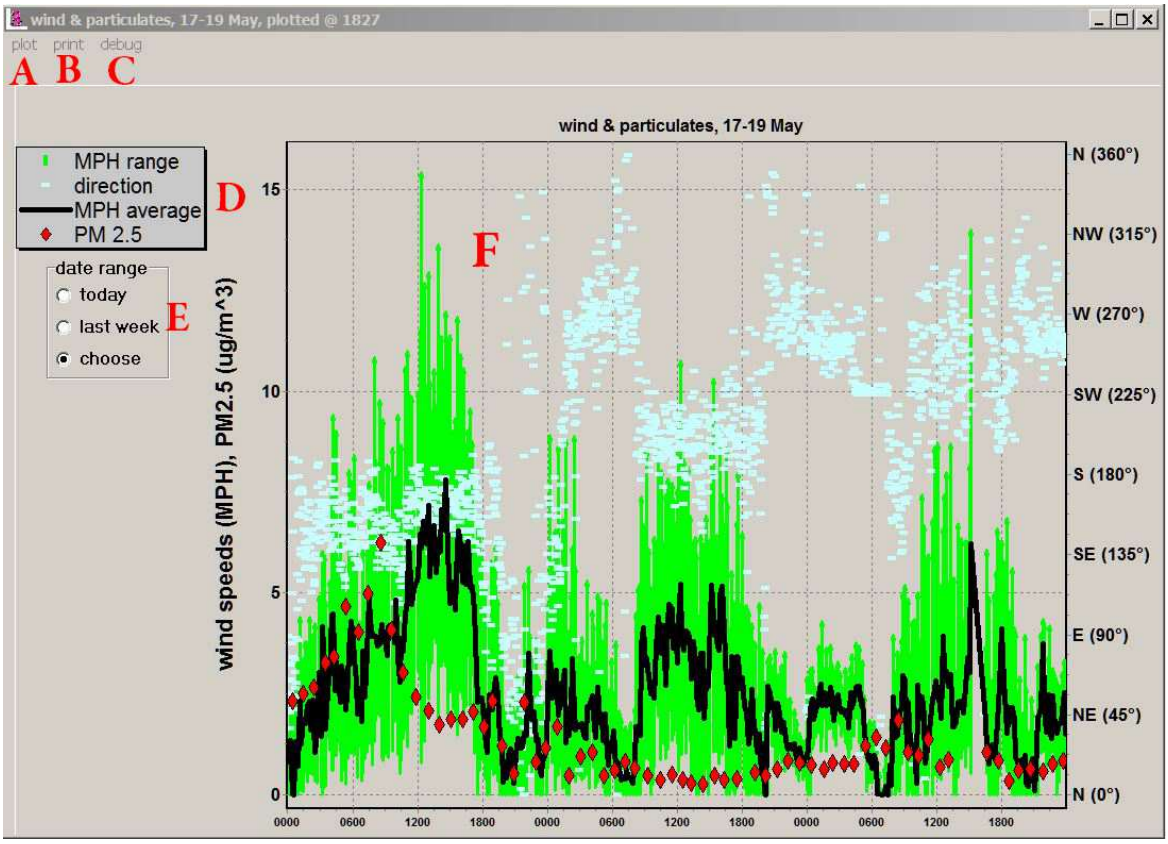


Figure 18, graph: wind

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 19, temperature ranges

9.10.2 user-defined graphs

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

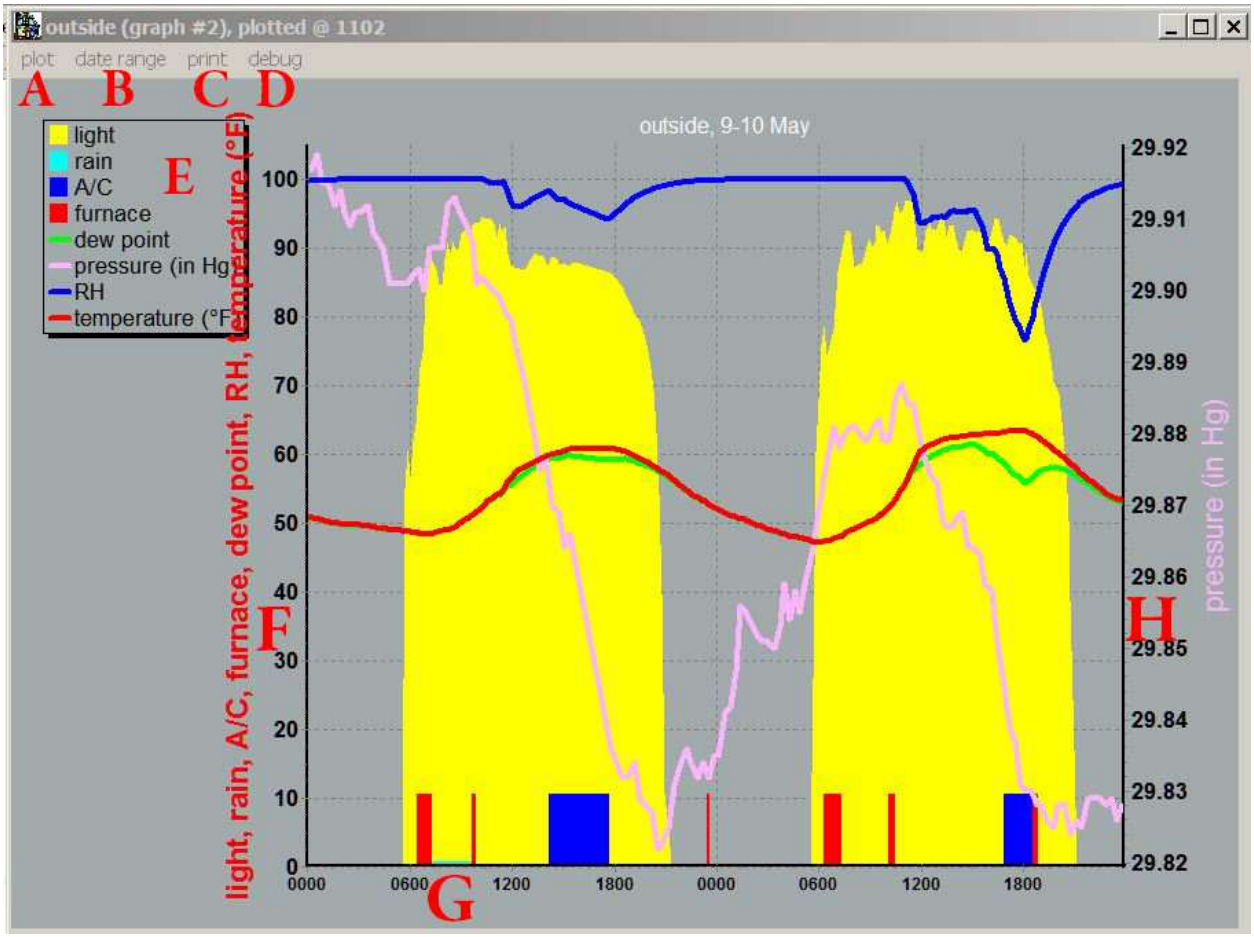


Figure 20, user-defoned graph

After this graph had been defined and named **outside**, by tools described below, it was accessed by clicking **graphs/outdoor/outside** on the main menu. It initially appeared showing 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 series** (temperature, relative humidity, and pressure) and **area series** (furnace, A/C, light, and rain[74]). A **point series** ($PM_{2.5}$) is visible above in Figure 18. Each **tWhatMeasured2**[75] value is permanently assigned a series type:

- **wm2DepthInches, wm2EventCount**[76], **wm2IsNowOn**, and **wm2Resistance**[77] are associated with area series;
- **wm2MassPM25** and **wm2Voltage** are associated with point series; and
- all other measurements are associated with line series.

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

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

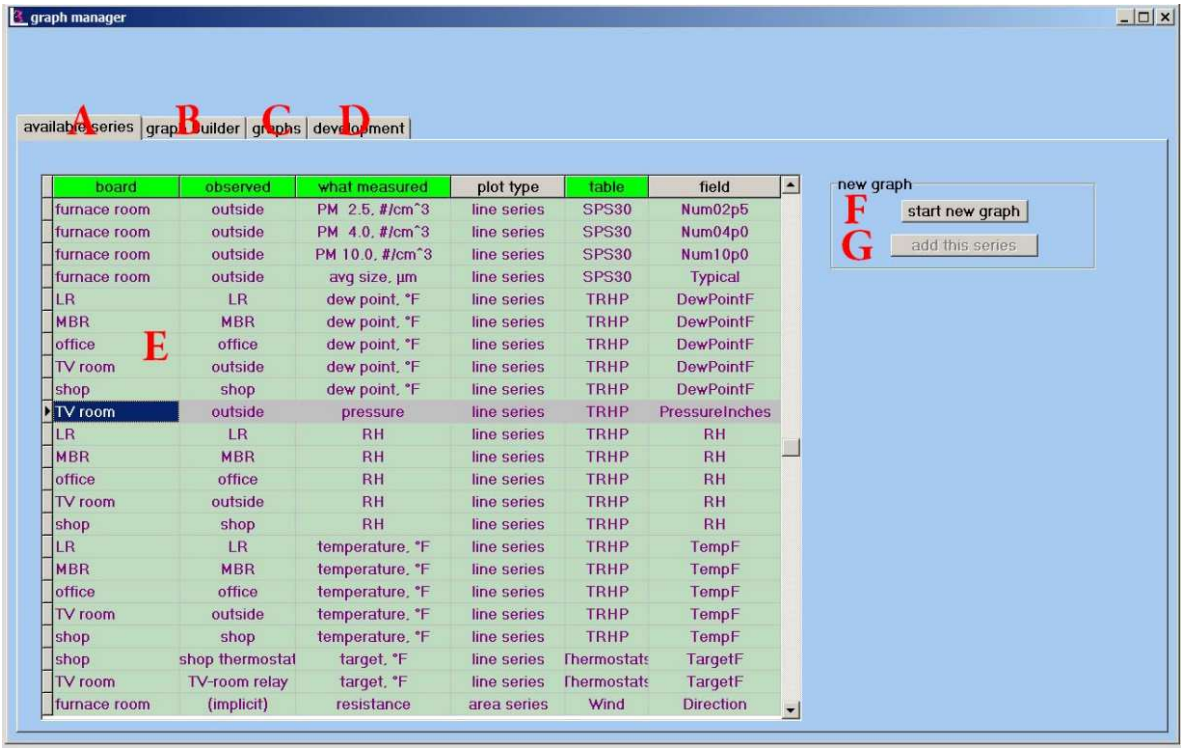


Figure 21, Graph Manager, first page

[74] There wasn't much rain on those days, but a cyan sliver is visible just above **G**.

[75] See Section 5.2.3.2 above.

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

[77] 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.

The first page of the Graph Manager presents (at **E**) a list of all the possible series. This page can be used to add a series to an existing graph, but I here demonstrate how it could be used to re-create the graph shown in Figure 20. The user would start a new graph (**F**) and go through the desired series, selecting each from the grid and then adding to the design by clicking **G**. It would then be time to move on (**B**) to the graph-builder page.

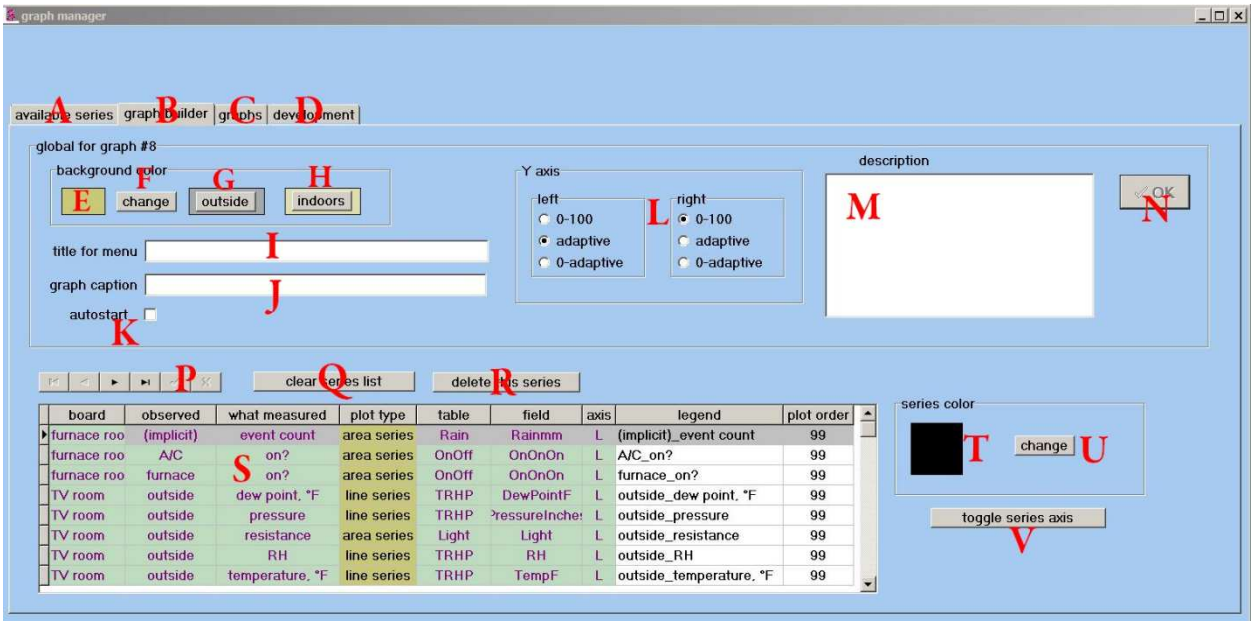


Figure 22, Graph Manager, graph builder

The edit box at **I** provides text that will be used in the **graphs** section of the main menu, and the contents of the edit box at **J** will be used as a caption for the graph on the screen. As soon as there is some text in each of these boxes, the **OK** button at **N** is enabled, and the graph could be saved and displayed.

The resulting display would be unattractive, to say the least.

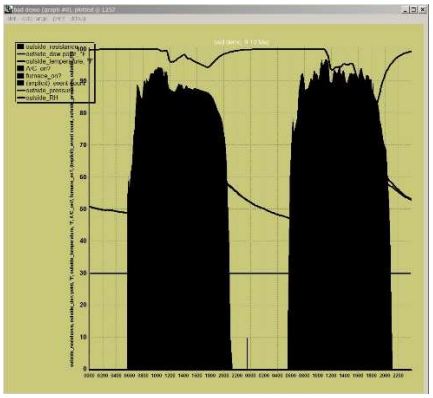


Figure 23, immature graph

The steps needed to turn this mess into a duplicate of Figure 20 are as follows:

- Enter **outside** in the edit boxes at **I** and **J**.

- Set a proper background color, either by choosing from the whole Windows gamut (button **F**) or by choosing one of the presets (button **G** or button **H**). The choice will appear in **E** and as the background color in the **plot type** column of **S**.
- Select the **outside/pressure** series in **S**, and click button **V** to change its Y-axis from left to right.
- Set up the left and right Y-axes, using the radio buttons at **L**. The choices are **0-100**, **adaptive** (running from just below to just above the observed data range), and **0-adaptive** (running from 0 to just above the observed data maximum). Here, the left axis should be **0-100** and the right axis (for the pressure series) should be **adaptive.**
- For each series, improve the automatically-generated text in the **legend** column of **S**. For example, change **outside_dew point, °F** to **dew point**.
- For each series, change the color with which it will be plotted. The button at **U** brings one to a Windows color-choosing dialog; the result is shown in **T** and as the text color in the **plot type** column of **S**.
- Revise the entries in the **plot order** column of **S**. Plot order comes into play when series overlap, because later-plotted series overwrite the pixels of earlier-plotted ones. In Figure 20, for example, the big yellow light series was evidently plotted first, since every other series overwrites it. The dew point and temperature were evidently plotted in that order, because at times of 100% relative humidity (that is, when dew point = temperature), the dew point line is hidden behind the temperature line.
- Optionally, enter descriptive text into the memo **M**.
- Optionally, indicate (box **K**) that this graph should be shown whenever the application runs.

Those steps would get the form to look like this

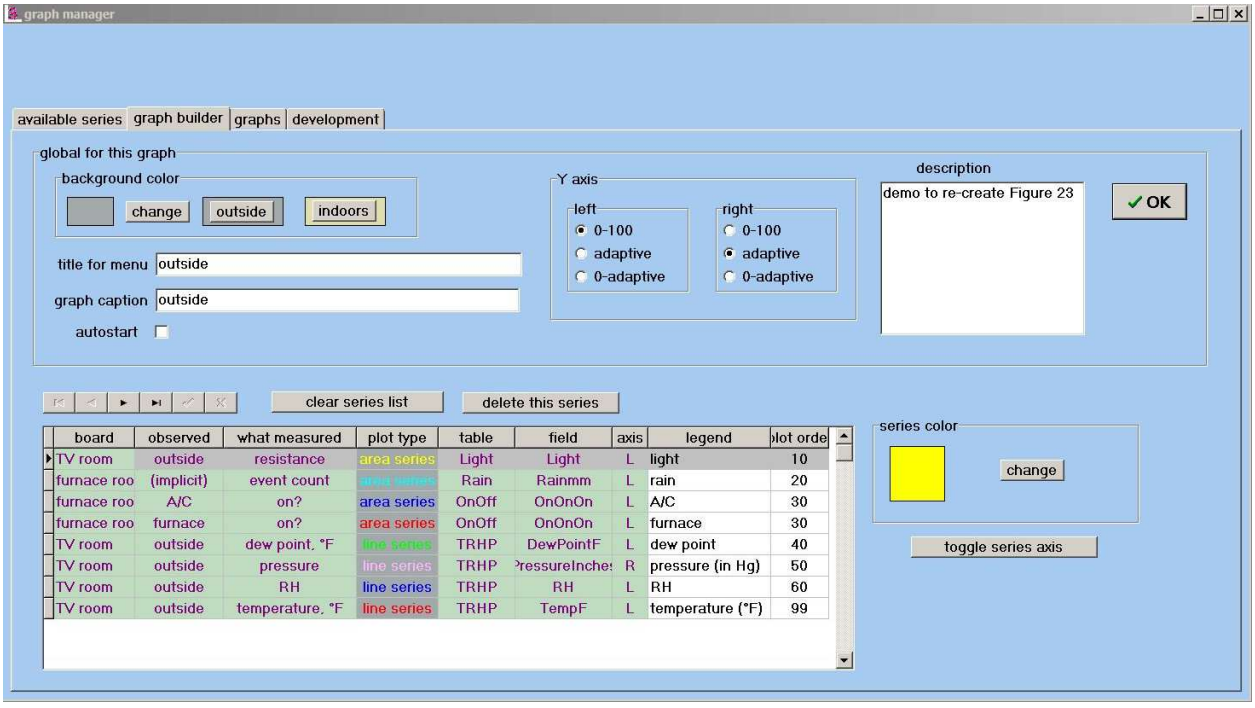


Figure 24, Figure 20 rebuilt

From there, the **OK** (button N in Figure 22) does the trick.

The third page (C) of the Graph Manager form has a grid listing all of the defined graphs. Double-clicking on the grid causes the data defining the selected graph to be loaded into the graph-builder page for editing.

9.11 miscellaneous

9.11.1 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.

9.11.2 Notes

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

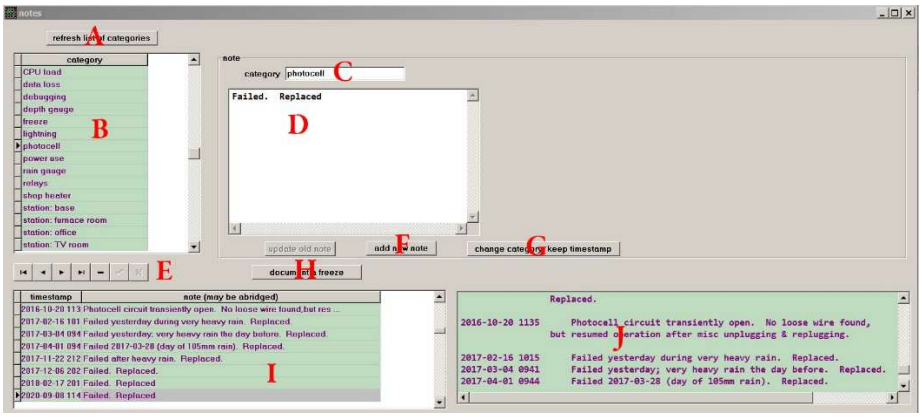


Figure 25, notes

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.

9.11.3 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.

9.11.4 clear warnings

A message becomes visible on the main form whenever a VC0706 camera at a peripheral station has detected motion. 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 (H in Figure 4) is garishly recolored, and the computer beeps. These alerts are suppressed by the **tweaks/clear warnings** item on the main menu.