Carma Weather Station:

Peter Teuben

Version 3.0

Updated: March 6, 2007

1 Functionality and Scope

1.1 Overview

The weather station provides the observatory with basic weather information (temperature, pressure, wind speed etc.), and publishes these (and some derived) variables to the standard CARMA monitor system. Several subsystems need this weather information for proper operation:

1. hazardous weather conditions, most notably wind speed. The fault system will be looking at these.
2. system temperature: the ambient temperature is important to obtain an accurate value for T_{sys}.
3. refraction: it is mostly the pressure that determines the amount of refraction (several arcmin at low elevation), and depending on required accuracy the temperature, humidity level and observing frequency are also needed.
4. delays: need a *good* atmospheric model, and again pressure, temperature and to some extent humidity are important here.
5. *should add something about required accuracy for some of these items*

Two identical weatherstations (a Campbell Scientific CR10X datalogger) have been in operation for a few years, one at SZA and one to sample data at the CARMA site (the circular buffer of the datalogger can contain many months of weather info at 5 minute samples). We plan to use this as our weatherstation (WS). In addition, there is a separate dew point sensor (DPS) for very accurate measurements of the ambient and dewpoint temperature (most simple weatherstations have questionable humidity sensors). Initial comparisons showed that the Relative Humidity given by the WS and computed from the DSP differ by a few percent, and the Dew Point Temperatures can be off by 1-2 degrees Celsius.

The opacity monitor (tipper) is part of another package.

1.2 Measured variables

1.2.1 WS variables

The following variables are read from the weather station, currently available every 5 minutes on station ID 222. Every 24 hours some averaged quantities are available on station ID 333, which we will ignore.

```
WX_STN_ID,     /* The station ID character (222/333) */
WX_YEAR,       /* The year number (eg. 1998) */
WX_DAY,        /* The day number (eg. 232) */
WX_TIME,       /* The time-of-day (eg. 1934 [ie. 19 hours, 34 minutes]) GMT */
WX_INT_TEMP,   /* The internal temperature of the weather station (C) */
WX_BATTERY,    /* The battery voltage (V) */
WX_AIR_TEMP,   /* The outside temperature (C) */
WX_HUMIDITY,   /* The relative humidity (%) */
WX_WIND_SPEED, /* The wind speed (m/s) */
WX_WIND_DIR,   /* The wind direction where the wind is coming from (degrees East of North) */
WX_PRESSURE,   /* The atmospheric pressure (mb) */
```

The WS is actually completely programmable. Depending on the modules one buys with it, and how the internal panel has been properly wired, various instruments can deliver variables. Thanks to Erik Leitch, some code is now available by which this can be done via Linux uploading new firmware. We expect to change the 5 minute information cycle down to whatever we need.

1.2.2 DPS variables

The DPS: returns only the ambient and dew point temperature, both in Fahrenheit.

1.3 A little weather background

At low altitudes our atmosphere has roughly an exponential scaleheight of 8.3km, thus the barometric pressure, to first order, can be given by as a function of the altitude as follows:

$$P(alt) \approx 1000 \quad e^{-alt[m]/8300}[mbar]$$

This formulae is actually used in the offline ephemeris if no weather information is available. The dew point temperature is defined as the temperature to which the air would have to cool (at constant pressure and constant water vapor content) in order to reach saturation. The relationship between Ambient temperature, Dew Point temperature and Relative Humidity have various approximations. We currently use the version by Hoffman & Welch from the BIMA code:

$$\frac{RH}{100} = e^{5370(\frac{1}{T_a} - \frac{1}{T_d})}$$

where temperatures are in Kelvin. Conversion routines using this relationship can be found in the class `carma::environment::Atmosphere`, though alternative formulae (e.g. Clausius-Clapeyron) are also available. [1]

1.4 Control Input

The weather station is an autonomous unit and does not need to be set into an operational mode. Hence no IDL code is needed.

1.5 Output

Software output consists of monitor data and occasional logging and tracing. The monitor points are described in detail in Section 2.2. Logging can occur if the weather station looses power (e.g. low voltage battery) Also, dangerous weather conditions should set an alarm and possible put the array into a "safe mode", however this is the responsibility of the fault system.

1.6 Administrative Summary

1.6.1 Estimated FTE Effort

Design:	0.25	
Implementation:	0.5	
Testing and integration:	0.5	(numbers are in man-months)
	—-	
TOTAL:	1.25	

[1] see also `http://www.faqs.org/faqs/meteorology/temp-dewpoint/`

1.6.2 Schedule

Package status:	open	
Work Package Analysis:	completed:	08/27/2002
Conceptual design:	completed:	n/a
Preliminary design:	completed:	02/03/2005
Critical design:		05/06/2005

2 Design

Monitoring functions for the weather station will be realized through a simple RS232 serial port. New code is needed for publishing the associated CORBA distributed object (DO) that external clients contact to activate its control functionality. An existing flexible serial port access routine (converted to C++) from the WASP project will be used to access the serial port.

2.1 Control API

The weather station has no controls, polling occurs on the RS232 port, though the firmware for both the DPS and WS can be enabled in an asynchronous mode.

2.2 Monitor Points

The following is the complete set of monitor points in the `carma.environment.weatherStation` hierarchy. All monitor points should be published at a 2 Hz rate.

- `ambientTemperature` (type: *float*; units: *C*)
  Outside ambient temperature. Measured with DPS or WS. See also weatherMode below.
- `dewpointTemperature` (type: *float*; units: *C*)
  Dewpoint temperature. Either measured (DPS) or derived (WS). See also weatherMode below.
- `humidity` (type: *float*; units: %)
  Relative Humidity, between 0 and 100%. Either measured (WS) or derived (DPS). See also weather-Mode below.
- `pressure` (type: *float*; units: *mbar*)
  Measured Barometric pressure.
- `windSpeed` (type: *float*; units: *m/s*)
  Measured Wind speed.
- `waterDensity` (type: *float*; units: g/m^3)
  Water vapor density
- `precipWater` (type: *float*; units: *mm*)
  Derived precipitable water vapor.
- `batteryVoltage` (type: *float*; units: *V*)
  Measured Voltage of the weather station battery.
- `weatherMode` (type: *integer*; units: *n/a*)
  Weather station mode. 1=WS only. 2=DPS only. 3=WS+DPS.

- `DewpointSensor.ambientTemperature` (type: *float*; units: *C*)
  Outside ambient temperature. Measured with DPS.
- `DewpointSensor.dewpointTemperature` (type: *float*; units: *C*)
  Dewpoint temperature. Measured with DPS.
- `WeatherStation.ambientTemperature` (type: *float*; units: *C*)
  Outside ambient temperature. Measured WS.
- `WeatherStation.humidity` (type: *float*; units: %)
  Relative Humidity, between 0 and 100%. Measured WS.

3 Dew Point Sensor

The program `wx` runs on `inyo` (a Sun) and connects to the WinNT (an old P5-75 Gateway) box and its output looks as follows:

```
Fast "Snapshot" Data
-------------------------------------------------------
Local OVRO Time/Date:       PST Mon Mar 14 19:30:03 2005
Universal Time/Date:        UTC Tue Mar 15 03:30:03 2005
Modified Julian Date:       53444.14587 days

Temperature, Ambient:         9.1 deg C    48.4 deg F
             Dewpoint:      -16.4 deg C     2.4 deg F

Barometeric Pressure:       880.6 millibars
Water Vapor Pressure:         1.7 millibars
            Density:          1.3 g/m^3

Columnar H2O Content*:        2.6 millimeters
Relative Humidity:           15 %
Wind Speed:                  11.7 mph     5.2 m/sec
     Direction:             311.5 deg    (NW)
     North Vector:            7.8 mph     3.5 m/sec
     East Vector:            -8.8 mph    -3.9 m/sec

*From local H2O density assuming 2 km scale height
-------------------------------------------------------
Current Generator Status

Fault              Warning            Status
(   )Low Oil Press (   )Low Oil Press ( X )Array on Util
(   )High Eng Temp (   )High Eng Temp ( X )Ctrl Bld Util
(   )Overspeed     (   )Low Eng Temp  (   )Generator Run
(   )Overcrank     (   )Low Fuel      (   )Array on Gen
(   )Not in Auto   (   )Fuel Tank Rpt (   )Ctrl Bld Gen
```

4 Implementation

User-level access to the weather system is provided by a C++ class named Weather, which contain the WS and DPS classes that directly talk to the serial port.

4.1 Code Reuse

SZA is using the same instrument (two identical systems were baught in year-????). Erik reported that in theory the firmware can be modified to have the weatherstation asynchronously send weather info at a pre-programmed rate (via firmware settings, not something we would want to control via IDL). This was never working according to advertised interfaces. However it's working fine by polling the serial port.
Some of the old code on the old OVRO WindowsNT3.5 based weather station box probably has some algorithms for the derived quantities that can be added to the current ones in `Atmosphere`.
WS up/downloading firmware code has been made available by Erik Leitch.

5 Programs

This package does not currently supply any programs.When the server runs, it locks up the serial ports and using even one of the test programs would result in crashing the server and/or producing bogus numbers.
`carma/environment/Test/tAtmosphere` : can be used to excersize the conversion routines between ambient temperature, dewpoint temperature and humidity.

6 Notes

- battery : it can also be monitored, returns battery voltage. does the new weatherstation have it?
- Paul tried a fiber-to-RS232 converter, which works fine for the DPS, but not for the WS.

Acknowledgements

The author wishes to thank Curt G, Steve Scott, Paul Daniel and Jim F(?) for help during the initial hardware integration and Chul Gwon for patiently explaining some CORBA internals for the examples.

7 Appendix A: Weather Station

The weather station is a CRX10 datalogger from Campbell Scientific[2] consisting of the following: a 3 meter tall UT3 tower, a Vaisala HMP35 temp and RH probe with solar radiation shield, a Vaisala CS105, PTB101B Barometer (600 to 1060 mbar), a 05103 RM Young Wind Monitor, a CR10X-1M Datalogger with 1Meg memory module and wiring panel, a 12 volt battery power supply with charge regulator, an MSX10 10 watt solar panel, and an ENC 16/18 Enclosure 16" x 18". A SC32B optical isolator is needed to converted the non-standard 9pin RS232 signal to a regular serial line, to enable the "normal" communication modes used on PCs.

7.1 Wiring

Currently a complex change through the required optical isolator to various forms of serial cables (9 pin, 25 pine, rj11).

7.2 Debugging

It is easy on serial lines to get your wires crossed, no pun intended. DCE, DTE, and remember RS232 cables are different from telecon RJ11(?) cables, and they've been used in various locations. See pictures on the weatherstation page[3]

7.2.1 USB serial

Once the `environment` has booted, the modules `keyspan` and `usbserial` should have been loaded, and via for example `dmesg` you should see sections such as:

```
...
usbserial.c: USB Serial support registered for Generic
usbserial.c: USB Serial Driver core v1.4
usbserial.c: USB Serial support registered for Keyspan - (without firmware)
usbserial.c: USB Serial support registered for Keyspan 1 port adapter
usbserial.c: USB Serial support registered for Keyspan 2 port adapter
usbserial.c: USB Serial support registered for Keyspan 4 port adapter
usbserial.c: Keyspan 4 port adapter converter detected
usbserial.c: Keyspan 4 port adapter converter now attached to ttyUSB0 (or usb/tts/0 for devfs)
usbserial.c: Keyspan 4 port adapter converter now attached to ttyUSB1 (or usb/tts/1 for devfs)
usbserial.c: Keyspan 4 port adapter converter now attached to ttyUSB2 (or usb/tts/2 for devfs)
usbserial.c: Keyspan 4 port adapter converter now attached to ttyUSB3 (or usb/tts/3 for devfs)
...
keyspan.c: v1.1.4:Keyspan USB to Serial Converter Driver
```

7.2.2 minicom

```
   Serial line:    9600 8N1: 9600 baud, 8 bit, no parity, 1 stop bit

   Set them manually in the file /etc/minirc.ws

# Machine-generated file - use "minicom -s" to change parameters.
pr port             /dev/ttyUSB0
pr bits             8
pr parity           N
pr stopbits         1
pu baudrate         9600
pu minit
pu mreset
```

[2] `http://www.campbellsci.com`

[3] current `http://www.astro.umd.edu/~teuben/carma/weather`

```
after which the command will be:


% minicom ws

assuming of course it has been stuck in the KEYSPAN #1 port!

You should (perhaps after hitting the RETURN key a few times) see a * prompt.
For debugging, the most important commands are:
   A   get the current status and memory counters
   B   backup a record
   C   get the time
   D   dump a record


*C
Y05 D0077  T19:06:18 C1294
*A
R+245610. F+573441. V3 A1 L+245610. E99 79 99 M1280   B+3.2132 C3336
*B
A1 L+245598 C0806
*D
01+0222.  02+2005.  03+0077.  04+1905.  05+17.75  06+18.87  07+0874.  08+2.854
09+1.098  10+164.2  11+40.66  12+0.000

A1 L+245610 C6312
*


??   should we see Online of Offline in minicom's status line at the botttom now ??

!!   this is an example of a good data record. The 'C' time (19:06:18) is less than
     5 minutes from the 4th data item in the 'D' line  (1905)
```

7.2.3 Command Mode

The weather station program actually runs every 3 seconds and puts the data into intermediate storage locations. To get to the intermediate storage locations, use minicom and do a carriage return or 2 to get the familiar * prompt. Then issue a **7H** (enter) to put the weather station into command mode. This can be verified by the prompt changing to a >. Then issue a ***6**. The weather station will respond with, MODE 06:0000. Use, Enter, to step through the intermediate locations or use, #N, where N is the number of the desired intermediate location. There are 28 intermediate locations of which only 19 are used by this program. The ones of interest to you are: 01 (Battery Volts, weather station power supply), 03 (Wind Speed in miles per hour), 04 (Wind Direction in degrees), 05 (Barometric Pressure in mbar), 06 (Air Temperature in degrees C), 07 (Relative Humidity in percent), 08 (Vapor Pressure of water in mbar), 09 (Saturation Vapor Pressure of water at the current air temperature in mbar), and 14 (Water Vapor Density in grams per cubic meter). The other intermediate locations used by this program are for intermediate calculation storage. To exit from command mode back into the 5 minute mode, let the WS idle for a short time (2.5 minutes found empirically).

```
*
*
*7H

>*6

MODE 06:0000
01:+13.901   voltage
02:+4145.5   program signature
03:+3.5803   wind speed (mph)
04:+299.73   wind dir (deg)
05:+786.43   pressure (mbar)
```

```
06:+8.4432   air temp (C)
07:+18.851   rel humidity (%)
08:+1.1048   vapor pressure of water (mbar)
09:+11.048   saturdation vapor pressure of water at current air temp (mbar)
10:+.18851   RH1 (intermediate calculation)
11:+2.0914   VPH20 (intermediate calculation)
12:+281.61   AIRTEMP1 (intermediate calculation)
13:+453.18   VPH20 (intermediate calculation)
14:+1.6092   water vapor density (g/m3)
15:+.52184   VP1 (intermediate calculation)
16:+2.0000   EXPON (intermediate calculation)
17:+.37276   VP2 (intermediate calculation)
18:+.09099   VP3 (intermediate calculation)
19:+.08613
20:+0.0000
21:+0.0000
...

*D
01+0222.  02+2007.  03+0004.  04+2050.  05+4.771  06+68.74  07+0774.  08+4.598
09+6.986  10+253.7  11+19.42

MODE 06:0000
01:+13.916
02:+4145.5
03:+8.1104
04:+245.42
05:+773.54
06:+4.8095
07:+68.772
08:+.86032
09:+8.6032
10:+.68772
11:+5.9166
12:+277.98
13:+1276.6
14:+4.5925
15:+.52184
16:+2.0000
17:+.37276
18:+.09099
19:+.08613
20:+0.0000
21:+0.0000
22:+0.0000
23:+0.0000
24:+0.0000
25:+0.0000
26:+0.0000
27:+0.0000
28:+0.0000
```

7.2.4 tWS

The test program `carma/environment/Test/tWS` can be used to debug communications:

```
   % cd $CARMA_BUILD
   % carma/environment/Test/tWS repeat=5
WS:   i=5 FC=337314686 Ta=15.66 Td=8.87881 RH=63.95 P=872 wSpeed=0.913 wDir=54.92 Batt=53.52 some=
WS:   i=4 FC=337314692 Ta=15.66 Td=8.87881 RH=63.95 P=872 wSpeed=0.913 wDir=54.92 Batt=53.52 some=
WS:   i=3 FC=337314697 Ta=15.66 Td=8.87881 RH=63.95 P=872 wSpeed=0.913 wDir=54.92 Batt=53.52 some=
WS:   i=2 FC=337314702 Ta=15.66 Td=8.87881 RH=63.95 P=872 wSpeed=0.913 wDir=54.92 Batt=53.52 some=
WS:   i=1 FC=337314708 Ta=15.66 Td=8.87881 RH=63.95 P=872 wSpeed=0.913 wDir=54.92 Batt=53.52 some=

   % carma/environment/Test/tWS traceLevel=1
{...  A: A R+400462. F+573441. V3 A1 L+400462. E99 99 99 M1280   B+3.2485 C3345  *}
{...  D: D A1 L+400462 C0814  *}
{...  B: B A1 L+400450 C0786  *}
{...  C: C Y05 D0126  T01:13:48 C1281  *}
{...  D: D01+0222.  02+2005.  03+0126.  04+0110.  05+15.75  06+63.58  07+0872.  08+08.53 09+0.888

    !! this is an example of a good (not ``flatlined'') data sample. The WS time is 01:13:48, and
       the last sample time is less then a 5 minutes from this (0110).
```

8 Appendix B: Dew Point Sensor

The Dew Point Hygrothermometer is a Model 1063 (aquired 1991 from Technical Services Laboratory[4], and returns the ambient temperature and a direct measurement of the dewpoint temperature from a chilled mirror hygrothermometer. It currently is configured in a mode where it returns the two temperatures (in 23 bytes) every 2 seconds accross a 600 baud serial line. This theoretically takes 0.345 seconds to transmit.

8.1 Wiring

Serial line at the DPS (currently on the roof of the OVRO control building) is converted to fiber, braught into the building and converted back to a serial line.

8.2 Debugging

8.2.1 minicom

```
   Serial line:    600 8N1: 600 baud, 8 bit, no parity, 1 stop bit

   Note that minicom doesn't support this speed from the menus, but you can
   edit them manually in the file /etc/minirc.dps

# Machine-generated file - use "minicom -s" to change parameters.
pr port             /dev/ttyUSB1
pr bits             8
pr parity           N
pr stopbits         1
pu baudrate         600
pu minit
pu mreset


   after which the command will be:


   % minicom dps

   assuming of course it has been stuck in the KEYSPAN #2 port!

   Since the DPS is an autonomous machine, you can't talk to it, it will
   spit out numbers every 2 seconds:

.T+ 58.3/D+ 18.6 P68.
.T+ 58.5/D+ 18.6 P6A.
.T+ 58.6/D+ 18.5 P6A.
.T+ 58.7/D+ 18.6 P6C.
.T+ 58.8/D+ 18.6 P6D.

   The 58.* values are the ambient temperature in Fahrenheit, where
   the 18.* values are the dew point temperature.  The P6* values are
   a checksum (which we ignore) to validate the two preceding numbers
   are ok. The T+ and D+ are equally so important, they indicate
   good values....
```

8.2.2 tDPS

The test program `carma/environment/Test/tDPS` can be used to debug communications:

[4] `http://www.tslinc.com/`

```
   % cd $CARMA_BUILD
   % carma/environment/Test/tDPS repeat=5
DPS:  i=5 FC=335733070 dFC=1 Ta=22.6667 Td=-4.44444 RH=16.0163
DPS:  i=4 FC=335733074 dFC=4 Ta=22.6111 Td=-4.44444 RH=16.071
DPS:  i=3 FC=335733078 dFC=4 Ta=22.6111 Td=-4.38889 RH=16.1375
DPS:  i=2 FC=335733082 dFC=4 Ta=22.5556 Td=-4.38889 RH=16.1926
DPS:  i=1 FC=335733086 dFC=4 Ta=22.5    Td=-4.38889 RH=16.248
```

where it should be noted that RH has been derived from Ta and Td, not measured. Note these are not the raw measured values, the instrument returns two temperatures in Fahrenheit, and have been converted to Celcius for readability here. Increasing traceLevel will show the raw measured values in the mostly readable output from the serial port:

```
   % carma/environment/Test/tDPS traceLevel=2
{.... buffer[23]="T+ 58.0/D+ 28.5 P65"}
{.... T+ offset by 1 bytes}
DPS:  i=1 FC=337314227 dFC=1 Ta=14.4444 Td=-1.94444 RH=32.3566
```

9 Appendix C: Linux cautionary notes

9.1 Setup

Given a standard CARMAlized linux workstation, the follow items need to be in place for the weather station:

- wider permission on `/dev/ttyUSB*`. We used `chmod 666 /dev/ttyUSB*` The permissions on a serial device may be reset upon a reboot, which is related to the new udev filesystem. The official udev way is to set those permissions in the file `/etc/udev/permissions.d/50-udev.permissions`.
- sudo more mortal users to use minicom: add the line `Cmnd_Alias MINICOM = /usr/bin/minicom` to the file `/etc/sudoers`. This will allow you to make `/etc/minirc.ws` and `/etc/minirc.dps` if you desire to debug with `minicom`.
- *anything else Paul?*

Since we are using the KEYSPAN 4port serial-USB converted a few cautionary words for systems that don't auto(un)load their modules so well. Once the KEYSPAN is plugged into the USB port, a **red light** should show up on the base right under the USB (open circle) symbol. This means the drivers were loaded. Once you connect a serial device, the moment the software opens the port a **green light** will be on at the port (1,2,3 or 4) that is being used. **Never disconnect the USB cable at this stage**, though it's safe to disconnect the serial device itself. If you would, the device would most likely lock up and a reboot is needed to clear the status.

9.2 Runtime

The weather station normally runs on `environment.carma.pvt` as the task `weatherStation`. Since it keeps the serial port to both DPS and WS open, running test programs like `tWS` and `tDPS` will either crash `weatherStation` (for `tWS`) or return nonsense (for `tDPS`), depending how the hardware deals with multiple attempts to access.

```
WS:  if a 2nd client comes in, 1st one dies and 2nd takes over
DPS: 2nd client just returns no data, need to stop the server

## on the server (normally acc.carma.pvt)


% /opt/rt/bin/imradmin --imr corba --get-server-info WeatherHost
% /opt/rt/bin/dumpMonitor frames=1 component=Weather


% /opt/rt/bin/imradmin --imr corba --stop-server WeatherHost
% /opt/rt/bin/imradmin --imr corba --reset-server WeatherHost

% /opt/rt/bin/imradmin --imr corba --start-server WeatherHost


## on the hostmachine (normally environment.carma.pvt):

% /opt/rt/bin/dumpMonitor frames=1 subsystem=Weather


% /opt/rt/bin/imradmin --imr corba --set-server WeatherHost args "imr=corba.carma.pvt:20000 traceI
```

```
## tinkering with the args or executable, good for debugging via your own

% /opt/rt/bin/imradmin --imr corba --get-server-info WeatherHost
% /opt/rt/bin/imradmin --imr corba --set-server WeatherHost args "imr=corba.carma.pvt:20000 traceF

% /opt/rt/bin/imradmin --imr corba --set-server WeatherHost exec /home/teuben/carma/build/bin/weat



Server WeatherHost:
    ID:                       15
    Status:                   stopped
    Name:                     WeatherHost
    Host:                     environment.carma.pvt
    Path:                     /opt/rt/bin/weatherStation
    RunDir:                   /home/control
    Arguments:                imr=corba.carma.pvt:20000 traceFile=syslog traceLevel=1 useDBMS=true
    Activation Mode:          shared
    POA Activation:           true
    Update timeout (ms):      500
    Failure timeout (secs):   15
    Maximum spawn count:      2
    Started manually:         no
    Number of times spawned:  0
```

9.3 Testing on a localhost

Just for completeness, here's an example annoted shell session of running a weather station server and monitor on an isolate localhost.

```
       # rc.carma will take a few seconds until the shell prompt returns
1% scripts/rc.carma --nosu --imr localhost --file imr/controlTestLocal.xml --dir /tmp/carma --noap

       # this will start carma_tools/bin/{imr,nameserv,notserv}
2% bin/frameScriberPublisher imr=localhost subsystem=weather &  ==  (client)
3% bin/frameCollator imr=localhost &    === acc (server)
4% bin/weatherStation imr=localhost sleep=1 &

       # subsystem= on the client, component= on the server
5% bin/dumpMonitor subsystem=Weather stats=t
6% bin/dumpMonitor subsystem=Weather frames=2

       # if ready to stop
7% scripts/rc.carma --nosu --imr localhost --file imr/controlTestLocal.xml --dir /tmp/carma --noap
```

9.4 Testing on environment

Here's an example annoted shell session of running the weather station server on `environment`, and being careful about not interfering with the system itself

```
       # rc.carma will take a few seconds until the shell prompt returns
       # notice the new non-conflicting port number (carma uses 20000) and FQN usage
       # if you do want to post monitor points to the system just use 20000 instead
1% scripts/rc.carma --nosu --imr environment.carma.pvt:30000 --file imr/controlTestLocal.xml --di
1% scripts/rc.carma --nosu --imr corba.carma.pvt:20000       --file imr/wx.xml --dir /tmp/carma --
```

```
       # this will have started carma_tools/bin/{imr,nameserv,notserv}

2% bin/frameScriberPublisher imr=environment.carma.pvt:30000 subsystem=weather &
3% bin/frameCollator imr=environment.carma.pvt:30000 &
4% bin/weatherStation imr=environment.carma.pvt:30000 sleep=1 emulate=false ws=true dps=false &
       # this will oddly (?) also publish them as monitor points, so you can monitor them with e.g

       # subsystem= on the client, component= on the server
5% bin/dumpMonitor subsystem=Weather stats=t
6% bin/dumpMonitor subsystem=Weather frames=2

       # if ready to stop
7% scripts/rc.carma --nosu --imr environment.carma.pvt:30000 --file imr/controlTestLocal.xml --di
```

10 Appendix D: Monitor Points

There is no good recipe for adding a new monitor point, but here are some points that certainly would have to be taken care of. Examples are taken from the weather station(s) code, a relatively simple CARMA component in itself:

1. Add the monitor point in the appropriate spot into `carma/monitor/WeatherSubsystem.mpml`
2. Add the appropriate item in e.g. `carma/environment/WS.h`, and grab it from the instrument in `carma/environment/WS.cc`
3. Add the item to `carma/environment/Weather.h` and grab it from WS in `carma/environment/Weather.cc`
4. grab and publish the value in the DO `carma/environment/weatherStation.cc` in something like `wss.weatherStation().`
5. If needed, modify `carma/ui/rtd/windows/rtdweather.cc`