Raspberry Pi Weather Station

Objective

The Raspberry Pi is an educational tool, encouraging young and old to get under the hood of technology to figure out what makes it tick. At a minimum to get the Pi Weather Station working users need to configure a working Linux distro on the Pi, download Java onto it, and edit configuration files. Not hard, but if your current technical experience is turning on a tablet and sending text messages from your phone this is a leap into a new world. More than a million Pi's have been sold; many have taken that leap. Bravo.

Design Goals

As the Pi is inexpensive and uses little electricity it seemed an ideal platform for a dedicated application. It started as the desire for an accurate clock (not part of the Pi Weather Station per se) and grew from there.

Why Java?

The most important reason is that I program in Java every day. I am no master (Java can take a lifetime to master), but competent. To me Java is the ideal heavy-duty business language. It is not for real-time systems, it is not for gaming, nor complex UI apps. It is for moving and managing large amounts of complex data in large scale environments. My hope is that a few of you that enjoy this application will want to make changes and improvements and learn to code in Java. The world needs good Java coders.

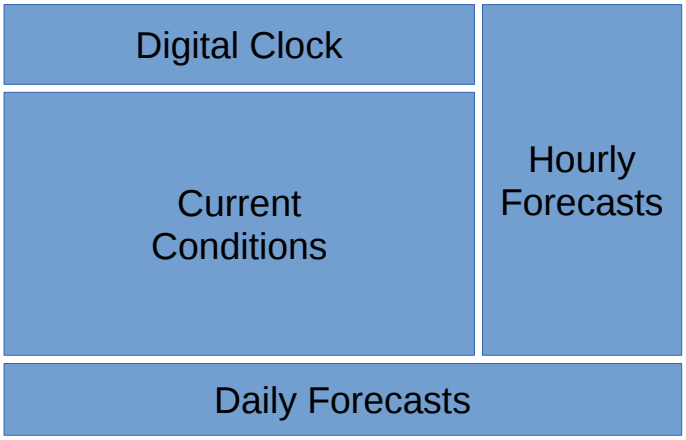
Encourage Responsibility

All the data for the Weather Station comes from weather.com that allows “developers” free access for infrequent requests, that is, less than 500 per day and 10 per minute. As it costs money to provide this service and weather.com does not have to give free access a core requirement was to limit the impact of the Pi Weather Station on weather.com. The app has a simple scheduling function, limiting the hours it requests updates. The default Home schedule is active Monday – Friday 6:00 a.m. to 9:00 a.m. and 6:00 p.m. to 11:00 p.m.; Saturday and Sunday 7:00 a.m. to 11:00 p.m. The default Office schedule is the ~inverse: Monday – Friday 7:00 a.m. to 6:00 p.m. and nothing the weekends. The app supports sharing the data within a network: a master station requests the data from weather.com and updates all registered slaves. If you have a station in the kitchen, one in the bedroom, and a third in the basement, it only requests data to a single station. Lastly as the simple scheduling is only hourly, I added a 30 second randomizer to the requests, that is rather than requesting the data exactly on the hour/minute, it waits up to 30 seconds to limit the number of simultaneous Pi Weather Station hits on weather.com. Not that I really expect thousands of these to spring up around the globe, but as a teaching tool I want to encourage thinking about the impact one person can have on the rest of the connected world.

Overview

Screen Layout

In its default layout the screen is divided into four areas, the digital clock, the current conditions, the hourly forecasts, and the daily forecasts.



The digital clock is provided by a separate program, dclock. The remaining three frames are provided by the Java program stored in the file PiWeatherStation.jar.

Data Source and Updates

The Current Conditions frame has three data feeds:

1. The current weather conditions showing the temperature and the text weather reports for the current day and night. By default this data is updated every 60 minutes.
2. The radar map updated every 20 minutes.
3. The “astronomy” data, i.e. the sun rise, sun set, and phase of the moon updated nightly

The Hourly Forecast shows a brief weather report for the coming hours (the first four in hourly increments, the next three in two hour increments, and the remainder in three hour increments. Default update is 30 minutes.

The Daily Forecast show brief weather reports for the coming days updated every 360 minutes (6 hours).

Installation

Prerequisites

A Raspberry Pi with Internet access running a Linux distro. All development was done with the standard “wheezy” distro, although as Java runs on most platforms it should run on almost anything. That being said, the instructions are based on wheezy.

I have it running on 256MB and 512MB Pi's without issue.

The user is expected to know enough Linux to be able to type in commands at a command prompt and

edit text files. A common text editor is nano, although I use vi.

raspi-config settings

With wheezy installed at the first boot you are taken to the raspi-config configuration menu. Using it, make the following changes:

- **Expand Filesystem** *(optional)*
  This expands the partition on the SD card to fill the available space.
- **Enable Boot to Desktop**
  Be sure to set to <No>. That maybe the default but I am not sure; it's easy to set.
- **Internationalization Options**
  Set your locale, timezone, and keyboard layout.
- **Enable SSH** *(optional)*
  An option found under the Advanced Options menu that allows you to log onto the pi from another computer.

Reboot the pi.

Update The System and Install Software Packages

Log in as root and run the two commands:

```
apt-get update
apt-get -y upgrade
```

to update the system, then:

```
apt-get install openjdk-7-jdk dclock x11-xserver-utils ntp
```

to install the required additional software packages.

Pi Weather Station Setup

Weather Data Feed Setup & Test

The first thing you need is a weather.com developer account. It is provided through wunderground (weather.com purchased them in 2012). Go to the login page at:

http://www.wunderground.com/weather/api/d/login.html

create a new account and get a key (look for a link to “Purchase Key”). Developer keys are free but you still have to follow the “Purchase Key” link. Sign up for the Anvil level that includes all the feeds the weather station needs. Your key will be 16 characters long, a mix of digits and letters from a-f (hexadecimal).

To test type the following URL into any browser substituting the values for your weather.com key, state (or country if you live outside the US), and city.

`http://api.wunderground.com/api/`*`WEATHER KEY`*`/conditions/q/`*`STATE`*`/`*`CITY`*`.json`

If you have everything correct a json file is displayed in the browser that begins like this:

```
{
        "response": {
                "version": "0.1"
                ,"termsofService":
"http://www.wunderground.com/weather/api/d/terms.html"
                ,"features": {
                "conditions": 1
                }
        }
```

If you do not, you cannot continue. You must have a valid weather.com key and know your state and city.

Invoke X Manually

X-Server is the core Linux software on which all graphical programs sit. Typically after booting, the next layer of software loaded is a window manager like Gnome, KDE, etc., that allows you to load programs and manage the windows in which they run. Being an appliance that does not require interaction, the weather station does not need a window manager. It does need the X-Server running.

To invoke X manually you need to create a text file in your home folder called:

```
.xinitrc
```

Yes, the filename begins with a dot, which makes it a hidden file. The command:

```
ls
```

will not show hidden files. Add `-a` to the command to show all the files in a folder. The contents of your .xinitrc for the weather station should be:

```
xsetroot -solid black &
#sets the background color black.
xset s off &
#disables screen blanking.

java -jar /home/pi/lib/Weather.jar -w WEATHER_KEY -s STATE -c CITY &
#starts java, telling it to run the program in Weather.jar with parameters.

/usr/bin/dclock -geom 700x175+210+10 -seconds -led_off black -bg black -bw 0
-fg gray -nofade -noblink -slope 90 -thickness .10
#displays the digital clock 70
# px (pixels) wide, 175 px high,
#starting 210 px from the left and 10 px from the top.
#The remaining parameters control color and the way the numbers are
dispalyed
```

substituting your values of `WEATHER_KEY`, `STATE`, and `CITY`. Once that file has been stored in your home folder, type the command:

```
startx
```

Automatic Starting

You must edit two files, /etc/inittab which is the file that controls how the Pi boots, and the .profile file

for the user you set as the Pi Weather Station user. For this example I assume you are using the default user **pi**.

Edit /etc/inittab

Load /etc/inittab in your favorite text editor with root privileges (either log in as root or precede your command with sudo, i.e. **`sudo nano /etc/inittab`**

Comment out the line below by adding # to the beginning of the line.

```
1:2345:respawn:/sbin/getty --noclear 38400 tty1
```

and insert this line:

```
1:2345:respawn:/bin/login -f pi tty1 </dev/tty1 >/dev/tty1 2>&1
```

This logs in the user pi on the first terminal.

Edit The Hidden File `/home/pi/.profile`

As noted earlier, hidden files start with a dot. To the bottom of the file add these lines:

```
ifconfig | grep "inet "
sleep 5
startx
```

The first two lines are optional. The first displays the IP address for the device which may be useful if you have not set it specifically in your DHCP server. You can use that to remotely log into the Pi with ssh and access the web page provided by the Pi Weather Station. The second waits for 5 seconds so you have time to see the IP address :-). The third line starts X.

Advanced Topics

Web Access to the Weather Station

Assuming your IP address is 192.168.1.100 in any browser on the same network as the Pi Weather Station enter the URL:

http://192.168.1.100:8080

From here you may refresh any or all the data feeds, review the schedules, and by following the Weather Station Settings link at the bottom of the page, all the settings.

Property File

Rather than entering the key, state, and city on the command line you may use a property file, that is, a file that has tuples (tech term for pairs of values, hey this is for education) of property names and values. For example:

```
state=NY
city=Pittsford
weather.key=1a2b3c4d5e6fab
```

Save this file anywhere and put it's location on the command line that starts java, i.e.

```
java -jar /home/pi/lib/PiWeatherStation.jar /home/pi/weather.properties &
```

The following table documents values that you can put in the property file to control the way the weather station performs.

Property	Explanation
weather.key	Your 16 character hexadecimal wunderground developer API key value.
state	Your state (US) or country.
city	Your city.
schedule	The default schedule for when the device downloads data from weather.com. Values are entered as hours. Afternoon hours may be entered with "p" suffix or on a 24 hour clock. Individual hours are comma separated, ranges are separated with a dash. Examples: **schedule=6-9,6p-11p** this schedules the first download for 6 a.m., continuing the downloads till 9 a.m., then waiting until 6 p.m. to restart downloads the continue to 11 p.m. **schedule=9-17** First download at 9 a.m. downloading through 5 p.m. There are four special values: **all** update every hour. **none** disable any updates. **home** set weekdays to **6-9,6p-11p** and weekends to **7a-11p**. **office** set weekdays to **7-6p** and disable weekends..
schedule.su	The Sunday override of the default schedule.
schedule.xx	The daily overrides for the days of the week, where xx represents the first two characters of the day, su, mo, tu, we, th, fr, sa. The format is the same as the default schedule.
conditions.refresh.interval	The number of minutes between refreshing of the conditions data. Default 60.
forecast.refresh.interval	The number of minutes between refreshing of the forecast data. Default 360.
hourly.refresh.interval	The number of minutes between refreshing of the hourly data. Default 30.
radar.refresh.interval	The number of minutes between refreshing of the radar map. Default 20.
conditions.location.x	The x dimension (across the screen) in pixels of the location of the conditions frame.
conditions.location.y	The y dimension location of the conditions frame.

Property	Explanation
conditions.width	The width of the conditions frame.
conditions.height	The height of the conditions frame.
conditions.radar.width	The width of the radar map on the conditions frame.
conditions.radar.height	The height of the radar map on the conditions frame.
hourly...	The location.x, location.y, width, height of the hourly frame.
forecast...	The location.x, location.y, width, height of the forecast frame.
forecast.column.count	The number of columns (days) of forecast data displayed.
hourly.row.count	The number of rows of hourly data displayed.
hourly.text.height	The font size for the hourly forecast data. Note: occasionally when there is a lot of hourly forecast data the display wigs out and shows nothing in the hourly frame. If it happens regularly you may want to lower either the row count or text height.
master.ip	This is the IP address of the master Pi Weather Station used in any slave stations. The slave station gets all the weather data from the master station limiting the number of data requests to weather.com

Master-Slave Setup

If truth be told, I was pleased with this idea and implementation. If you have multiple stations in your home or office, why duplicate weather.com downloads? So I implemented a classic Java listener pattern. Slave stations know the IP address of the master and using standard web communications register themselves with the master. When the master updates one of the feeds it iterates over the list of slaves that have registered and tells them to update themselves, which they do by loading the updated feed file from the master station. Say the master is at 192.168.1.100 and a slave at 192.168.1.101 wants to get feeds from that master. Here are the communications that occur:

- Slave registers with master:

  ```
  http://192.168.1.100:8080/registerListener
  ```

  The master gets the slave's IP address from that call and stores it in a collection (Set). The slave sets the feed URLs (i.e. the location of the feed files) to the master station:

  ```
  http://192.168.1.100:8080/get/astronomy.json
  http://192.168.1.100:8080/get/conditions.json
  http://192.168.1.100:8080/get/radar.gif
  http://192.168.1.100:8080/get/forecase.json
  http://192.168.1.100:8080/get/hourly.json
  ```

- When the master updates feeds it sends one of the following communications to the slave:

  ```
  http://192.168.1.101:8080/updateEvent/astronomy
  http://192.168.1.101:8080/updateEvent/conditions
  http://192.168.1.101:8080/updateEvent/radar
  http://192.168.1.101:8080/updateEvent/forecast
  http://192.168.1.101:8080/updateEvent/hourly
  ```

  telling the slave to update the named feed, which was set to get from the master when it

registered.

If there are multiple slaves each registers with the master an update event is sent to each of the slaves.