JP1.10

CROSSING PLATFORMS: PCS AND AWIPS USING THE PYTHON PROGRAMMING LANGUAGE

C. Michael Callahan
National Weather Service, Louisville KY

1. INTRODUCTION

The author has worked with a number of different computer systems over the years, and has re-written many programs. With the arrival of AWIPS, he decided to investigate cross-platform programming. He experimented with a number of different languages but finally settled on the Python programming language. Python was created to be an interactive, dynamic, object-oriented, cross-platform language (van Rossum 2000a). Because it is free and widely available, it has been used in many different applications, ranging from small system scripts to large, complex Web-based systems. The author found the language attractive because of its power and ease of use. Also, Python is not dogmatic. It does not force one into a certain programming style. For example, while Python has excellent object-oriented features, one can write in a procedural style just as easily. After learning a few new concepts and Python's simple syntax, the programmer can quickly adopt old algorithms to Python, and then improve these programs as they become familiar to the more powerful features of the language. The author now uses Python for most of his programming tasks.

The standard library that comes with Python is a rich collection of classes and functions that makes development easy (van Rossum 2000b). Included are interfaces to FTP, HTTP, SMTP, XML, CGI, and other Web technologies.

Python includes a module which links to the Tk library in order to create graphical user interfaces (GUIs). Using this module, *Tkinter*, one can create Python programs that adopt the native look-and-feel of any platform (Lundh 2000). Tk is included on almost all Unix-based computers, but rarely is found on computers running Windows. In order to eliminate this potential problem, the standard Python installation package for Windows also installs TCL and Tk under the Python directory. Thus, by using Tkinter, one can write GUI programs that will run anywhere there is a Python installation.

* *Corresponding author address:* C. Michael Callahan, National Weather Service, 6201 Theiler Lane, Louisville, KY 40229; e-mail: mike.callahan@noaa.gov

In order to show the power of the Python programming language, this paper will describe two applications, *Dial* and *Convert*. Dial is a Windows-based program which was designed to interrogate hydrologic gauges, and send their data to the hydrologic AWIPS database. Convert is a simple GUI-based meteorological converter whose code runs on many platforms.

2. DIAL

The Local Data Acquisition and Dissemination (LDAD) component of AWIPS originally was envisioned to accomplish what Dial does. However, the LDAD hardware is almost overloaded with background processes and the gauge dialing software has been difficult to set up and maintain. The author felt that the gauge dialing task would be better accomplished using a PC and not tying up valuable LDAD resources. The PC interrogates the gauges, encodes the collected data into the Standard Hydrometeorologic Exchange Format or SHEF (NWS 1998), and then sends the data to LDAD, which allows AWIPS to post the data into its hydrologic database. In a sense, AWIPS hands the interrogation task to the PC, rather than the LDAD hardware. However, with a few changes dealing with modem communications, Dial could run on LDAD or other hardware as well.

The program is structured across five files whose combined size is about 114K. The main code is contained in DIAL.PY which handles the interfaces, error trapping, interrogation, decoding, and SHEF encoding. Currently, Dial supports 4 types of gauges: Handar 550A (known in the NWS as a LARC), Handar 560A, Campbell Recorder 10, and Sutron 8200. The modular design makes it easy to add new gauge types for future expansion.

The simple interface to the modem is in SMODEM.PY. This is the only file which is specific to Windows. In order for Dial to run on another platform, it would be necessary to rewrite SMODEM.PY. However, this task would not be too difficult.

The AWIPS LDAD interface is contained in two files, AWIPS.PY and AWIPARMS.PY. The first file is a simple interface to send data to AWIPS via LDAD. The transmission is through a mapped network drive and/or

FTP. AWIPARMS.PY contains the parameters for a mapped network drive and FTP servers of up to three different LDADs.

Most of the information Dial needs is contained in DIAL.INI. The old INI style file was chosen rather than to use the Windows registry. First of all, INI files are easy to maintain and even re-create in case of a failure. Also, INI files are portable.

Dial can be executed interactively or in an automatic mode by using command line parameters. The former method generally is used during high water events to obtain the most recent data. The latter method is used routinely when Dial is started by the Windows 98 task scheduler. Thus, one program handles routine and special data collection.

2.1 Dial Program Structure

DIAL.PY was written as an object-oriented program. Below is a list of the classes created:

InitStream: Encapsulates the routines needed to read the ini file.

Catcher: Catches any errors that arise while the GUI is running.

Gui: Encapsulates not only the GUI but all the events that the GUI can generate. Most of the code for the program is contained in this class.

SMODEM.PY creates these classes:

Timer: Encapsulates a very simple stopwatch-type timer.

Modem: Encapsulates a simple modem interface for Windows PCs.

AWIPS.PY creates this class:

Ldad: Encapsulates an interface for sending products to LDAD.

2.2 Dial Operation

Dial first opens the log file, DIAL.LOG, routes all future output to the file, and writes the start message. Then, it checks the command line.

If it finds two options, it sets the mode to automatic and assumes the first option is a gauge list. The program tries to open the gauge list and read the gauges into memory. If this fails, Dial assumes the first option is the ID of a gauge and puts this into memory. The second option is stored as the hours of past data desired.

If no options are found, the mode is set to interactive, the gauge list is cleared, and hours desired is defaulted to 6.

Now, the initialization file, DIAL.INI, is read, checked, and the data stored into memory. Next, the modem port is initialized and the modem is reset and tested. The modem must respond with "OK" within two tries or else the program logs the error and exits.

Now, the program starts Tkinter, creates the GUI, and begins processing all user-generated events (Fig. 1). After the user selects the desired gauges and clicks on the DIAL command button, or the program is in automatic mode, the gauges are placed into a call list and the dialing process begins.

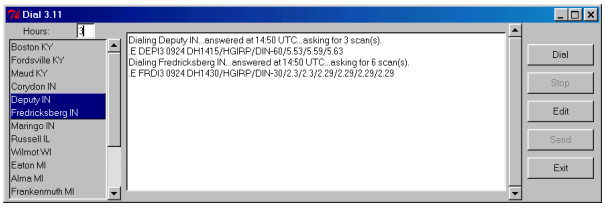

Figure 1: *Dial GUI*

For each gauge, the modem port is set to the correct parameters, and a message is written to the log stating that the dialing process has begun. The gauge is dialed and the program waits for the gauge to answer. If there is no dialtone, the line is busy, or there is no answer after two tries, the failure is written to the log and the next gauge is called. If the gauge does answer, the gauge is polled and the raw data it sends back is stored and written to the log file. Each gauge type has its own polling and processing routines.

Once the polling is complete, the processing begins. The raw data is converted to a SHEF format report, then stored and written to the log (Fig. 2) and displayed on the screen. The polling and processing continues until all gauges are called, or the user clicks on STOP. Clicking STOP while in the automatic mode stops dialing and switches the program to the interactive mode.

At this point in the interactive mode, the user can call more gauges. When the dialing is completed, he or she clicks on EDIT. At this point, the screen is cleared and the SHEF messages of all the gauges are displayed. The user can edit the messages to add comments or correct bad data.

When the editing is completed, the user clicks on SEND. The program will collect all the SHEF messages

```
>>>Dial 3.10 started Thu Sep 20 12:58:48 2001 UTC.
Dialing Gills Rock WI... No dialtone.
Exiting... stopped Thu Sep 20 13:04:52 2001 UTC.

>>>Dial 3.10 started Thu Sep 20 14:41:04 2001 UTC.
Dialing Deputy IN... answered at 14:42 UTC... asking for 3 scan(s).
6
12:15 07.08 07.08
13:15 06.99 06.99
14:15 06.88 06.88

.E DEPI3 0920 DH1415/HGIRP/DIN-60/6.88/6.99/7.08
Dialing Fredricksberg IN... answered at 14:42 UTC... asking for 6 scan(s).
12
12:00 02.39 02.39
12:30 02.41 02.41
13:00 02.42 02.42
13:30 02.43 02.43
14:00 02.44 02.44
14:30 02.44 02.44

.E FRDI3 0920 DH1430/HGIRP/DIN-30/2.44/2.44/2.43/2.42/2.41/2.39
Sending SHEF data to AWIPS... using FTP... First attempt was successful.
Exiting... stopped Thu Sep 20 14:44:15 2001 UTC.
```

Figure 2: *Dial log*

and write them to a temporary file, DIAL.SHF. Next, it will add the correct headers and trailers to the file. Now, it attempts to copy DIAL.SHF to a network drive which points to the local LDAD */data/Incoming* directory. The program now looks at the LDAD directory every 5 seconds for a user-determined amount of time to see if AWIPS ingests it and deletes the file.

If this fails, the program will try to logon the FTP server of the local LDAD and send the file to */data/Incoming*. Again, it waits to see if AWIPS ingests and deletes the file. If this fails, two other FTP servers of remote LDADs are attempted.

If any of these FTP transfers fails, the program creates an email message and sends it to a list of recipients. This list, the frequency of transmitted messages, and the error messages desired is defined by the user. For example, the service hydrologist may want to know about failures every hour, but the system analysis may want to be informed about failures every 6 hours. Having three sites ensures that the decoded SHEF data will make it onto the AWIPS network, unless the PC's Internet connection fails. Notice, Dial will collect and disseminate data even if the local AWIPS is down.

After the program exits, the modem is reset, the modem port is released, the GUI is cleared, a closing message is written to the log, and the log is trimmed to delete the earliest session.

3. CONVERT

Convert is a cross-platform meteorological units converter that can handle 50 different conversions. While a units converter is nothing new, the author selected this project because he wanted to develop a completely cross-platform GUI program that was useful for all people in the National Weather Service. He also wanted to demonstrate how easy it was to accomplish this using Python.

Convert handles these conversions between these units:

Temperature:	°F, °C, °K
Length:	in, ft, mm, m
Speed:	mph, kt, km/h, m/s (note: these can also be used to convert between mi, nm, and km)
Pressure:	mb, mm of Hg, in of Hg
Relative Humidity:	temperature and dewpoint in °F, °C, °K
Heat Index:	temperature in °F, °C, °K; dewpoint in °F, °C, °K; or relative humidity in percent
Wind Chill:	(both the old and new equations) temperature in °F, °C, °K; wind speed in mph, kt, km/h, m/s

Convert was designed to be easy to install. All that is required is to copy the CONVERT.PY file to a directory available to all users and create links to it. No registry editing or start up scripts are required. The file is small, only 11K.

3.1 Convert Operation

After the user starts the program, it looks for a command line argument. If found, it sets the operating system DISPLAY environment variable to the argument. Windows and most Linux systems ignore this variable, but AWIPS requires it. This allows the display code to be cross-platform. The AWIPS display environment variable

for a text workstation is this: `xt#-xxx:0.0`, where # is the workstation number and `xxx` is the site. Graphics workstations are this: `ws#-xxx:0.0` for the left screen and `ws#-xxx:0.1` for the right screen.

Next, the program starts Tkinter and builds the GUI (Fig. 3a and 3b). Now it waits for the user to generate events. The program comes up ready to convert °C to °F. The user clicks on the desired input and output units to set up the converter. Note, calculating relative humidity, wind chill, or heat index requires two input boxes.

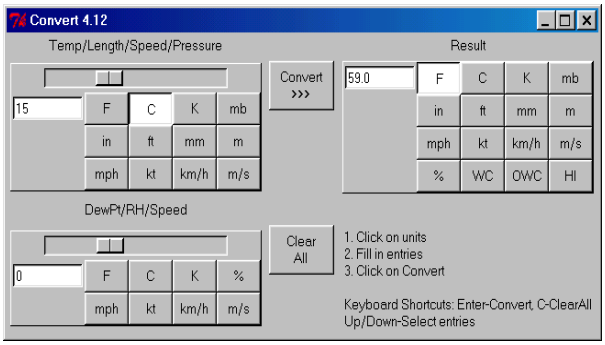

Figure 3a: *Convert GUI (Windows 98)*

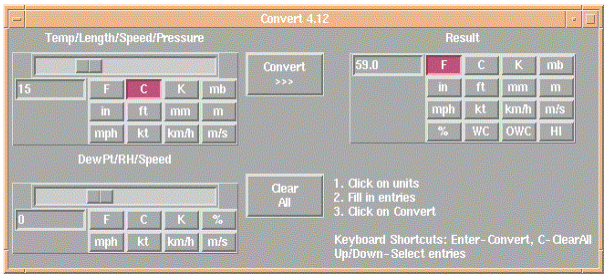

Figure 3b: *Convert GUI (AWIPS)*

Users can either use the mouse and click on the slider bar, or the keyboard to work the converter. For making many different conversions, the mouse and slider bar work best. When the user selects an input unit, pre-defined limits automatically are imposed on the slider. Slider values are always whole numbers.

To exceed these limits, or to make conversions without the mouse, the keyboard can be used. Once the converter is set up, the user presses the up and down keys to clear the top and bottom entry windows. The user then types in the values to convert and presses Enter.

4. SUMMARY

In the Louisville NWS office, Dial is used as an integral part of the Advanced Hydrologic Predication System (AHPS). Dial calls gauges every 6 hours, and the AHPS software produces hydrographs which are placed on the Web. This is the only source of real-time data for some gauges. Convert is used routinely, running on Windows PCs and AWIPS. Both Dial and Convert are being used in a number of different offices as well.

As hardware in NWS offices continue to evolve, it will be necessary to port operational programs to different platforms. Taking advantage of cross-platform techniques demonstrated in Dial and Convert using the Python programming language can make this process almost painless.

Python is not a programming panacea. It should not be used for system critical tasks where processing time is the most important factor. However, even in these applications one can write time-critical routines in C, and call them from inside Python applications.

In the past, the author has written programs and scripts in C, C++, C shell, Delphi, and Perl for many different computers and operating systems. Now, he finds that he can stay with one language, Python, not worrying about the platform, and get the job done.

5. REFERENCES

Lundh, F., 1999: An Introduction to Tkinter, www.pythonware.com/library/tkinter/introduction/index.htm

National Weather Service, 1998: Standard Hydrometeorological Exchange Format, Version 1.3, Weather Service Hydrology Handbook No. 1

van Rossum, G., 2000a: Python Tutorial, www.python.org/doc/2.0/tut/tut.html

___, 2000b: Python Library Reference, www.python.org/doc/2.0/lib/lib.html

6. ACKNOWLEDGMENTS

The author would like to thank the service hydrologists in the National Weather Service for testing Dial and Convert and for making excellent suggestions to improve the programs. He would also like to thank Ted Funk for reviewing this manuscript.