SmartPlug Technical

Python Smart Plug Technical

Contents [hide]

1 Overview
2 Components and Functions
2.1 The Controllers for Smart Plug
2.2 Global Values
2.3 Threads
2.3.1 The Tkinker Thread
2.3.2 HelperThread
2.4 Parameters
2.5 Polling
2.6 Logging
2.7 Other Classes
3 Coding Conventions
4 See Also
4.1 A Related Python Program

Overview [edit]

These notes are here so you can more easily modify the code. Contact me Russ Hensel if you would like additional information.

Before modifying the code it is best to understand how it works. Here is an overview of the general plan, details can be filled out by reading the code.

Currently the "Application" is really two different applications, each with its own main program and gui, but sharing a lot of the other code. I may merge them latter into one application. The two applications are: SmartPlugGraph (in smart_plug_graph.py) and SmartPlug (in smart_plug.py). The apps are very similar so I will explain them in a combined document: generally what is said about SmartPlug also applies to SmartPlugGraph.

The overall architecture is called the model view controller or MVC. The class SmartPlug can be viewed as the main class. To run the program run its file smart_plug.py (see code at end of file). SmartPlug is the controller in MVC it is responsible for all overall control, it directly or indirectly creates all other program objects.

The view component is called GUI (in gui.py and for SmartPlugGraph is called also GUI but is in gui_for_graph.py). It creates all the visible components, and relays user input to the controller. You can unplug the GUI object from the application and plug in new components. Don't like the GUI? You could modify mine, or you could make a modification and choose which one to use. This is sort of like a skin for an application.

Two other important components are called Logger (in logger.py) and Parameters (in parameters.py). The controller creates one of each, and make them available to the other components. The other components can interact with them, and uses them respectively for logging events, and getting access to parameters (those aspects of the application that are particularly easy to change). I describe more of this in My Python Coding Conventions

The application has a main thread running in a Tkinter mainloop. There is also a second thread called a "helper" running which makes some processing much easier. To make gui mainloop responsive to both the GUI and its own processing it uses a pseudo event loop or a polling subroutine that is implemented in xxxxx. The frequency which polling occurs is set in parameters, the relatively low rate of 100 ms between calls (.1 sec) seems to give a perfectly responsive application in most cases. I have run it as fast as once every 10 ms. Have not tried to find a limit.

Components and Functions [edit]

- Tkinter for the GUI
- pySerial for serial communications
- pyLogging for logging
- SmartTerminal a model-view-controller controller with some modifications (**Model–view–controller - Wikipedia** or perhaps closer to **Model–view–presenter - Wikipedia**)
- Second thread for processing without being blocked by/blocking the GUI
- Global Singleton for app cohesion.

The Controllers for Smart Plug [edit]

This is the class you create to run the application see the code at the bottom of the file, just run the file. Similar code is at the bottom of some of the other source files to make it convenient to run from those files, this is just to make it easier in development, the code in smart_terminal is the model for what should be used. Sometimes the code at the bottom of the file may have code for testing objects in the file, it may not be maintained, may not work as intended.

The __init__ method is the initialization and "run" method for the application. Much of the code that would normally be in it has been moved to .restart which is used to restart the application when just the parameters have been changed. See the docstring there.

Global Values [edit]

Yes I know that globals are bad, but they can be useful. For example many class instances need to access the parameter file. This can be done using the singleton class AppGlobal. It has values at the class level (not instance) that are available simply by importing the class. Most values are defaulted to None, and are set to valid values as the application initializes.

Threads [edit]

The application runs two threads, one for the GUI and one where processing can occur (HelperThread) with out locking up the GUI. There are 2 queues that allow the threads to communicate.

The Tkinker Thread [edit]

Once Tkinker is started it runs its own mainloop. In order to receive however we need to check the rs232 port from time to time. This is done in SmartTerminal.polling() I call this thread the GUI thread or gt. The terminal is configured to have a second thread called the Helper Thread or ht. In some cases the receiving of data is passed to the Helper Thread so where data is processed in addition to being posted to the GUI. See the section HelperThread for more info.

HelperThread [edit]

Refers to similar but different program !!

HelperThread in smart_terminal_helper.py This class provides the support for a second thread of execution that does not block the main thread being run by Tinker. I call the two threads the GUI Thread (gt) and the Helper Thread (ht). It can get confusing keeping track of which method is running in which thread, I sometimes annotate them with gt and ht. The helper thread is started by running HelperThread.run() which pretty much just runs a polling task in HelperThread.polling(). HelperThread.polling() is an infinite loop, it uses sleep to set the polling rate. When used with the green house processing module, it may call a function there that is its own infinite loop. There are a lot of details here, I should write some more about it.

Parameters [edit]

Parameters (in parameters.py) this is pretty much a structure (that is all instance variables) that is instantiated early in the life of the application. It passes values, strings, numbers, objects to application elements that need them. The instance of parameters is made globally available thru the AppGlobal class. Values in Parameters are treated as constants, read only. Much of the appearance and behavior of the application is controlled here.

The standard gui has a button to kick off editing of this file, the application may then be restarted (another button) with the new values.

There are a couple of meta parameters, the most important of which is mode which is then used in conditionals later in parameters. Except for this sort of thing there is really not much "code" in parameters. You can change this code pretty much as much as you like, as long as you end up setting up values for the required parameters.

The code is extensively commented: use that for documentation.

Values are often set to None as a default, then later set to some other value. Or the value may be set several times in a row (this is an artifact of messing with the values); only the last value set has any meaning.

For more info see: Configuration Files For Python

Polling [edit]

Both threads have method that perform polling for events often for items in their queue that may have been sent from the other thread. Info on a similar app in Python Smart Terminal Technical Details.

Logging [edit]

This uses the standard Python logging class. Logging level and other logging details are set up using the parameter file.

Other Classes [edit]

For now the documentation, as far as it exists is in the source code.

See: My Python Coding Conventions

Coding Conventions [edit]

See: My Python Coding Conventions

See Also [edit]

Master Page for this Project: Python Control of Smart Plugs

A Related Python Program [edit]

- Python Smart Terminal Technical Details More detailed technical issues.
- Smart Terminal GUI how the gui works and interfaces the user.
- Smart Terminal Convert from 2.7 to 3.6 converting a moderate size application.
- and the categories below may be useful (click on them). [category:Python]]

Category: Python SmartPlug

Content is available under Creative Commons Attribution Share Alike.