navigation
- Main page
- Community portal
- Current events
- Recent changes
- Random page
- Help

search
Go Search

toolbox
- What links here
- Related changes
- Upload file
- Special pages
- Printable version
- Permanent link

SmartPlug Help File

This is the Help "file" for the Python Application whose main page is: ***Python Control of Smart Plugs - OpenCircuits**

Help file for smart plug application (Ver4)

Contents [hide]

Application Features [edit]

- Works with multiple smartplugs.
- Provides on, off, timing, and energy recording functions from a graphical user interface (gui)
- Supports graphing of data. Currently power and energy over time. Multiple devices on one graph.
- Supports csv output of data.
- Saves data to a sql lite database that you can use any way you wish.
- Highly configurable.

Two Applications [edit]

This program is actually 2 main programs (I may merge them later). The first program is for control of the smartplugs and capture of data, the second for the graphing and processing of the data. The main routines may be run by respectively running: smart_plug.py graph_smart_plug.py I think (but have not tested) they may be run at the same time.

Download and Install [edit]

There is really no install program. Currently the application is intended for those who have at least a little familiarity with Python coding and can just insert the downloaded code into their development environment and run it. Code at **russ-hensel/Smart_plug Application** For most of you it will not run due to dependency problems. You will need to add them. Pretty much read the error messages and install the missing code. I use Anaconda Spyder so my preferred tool is conda (conda install pyHS100). If conda does not work or you do not use it use pip. pyHS100, a library for smart plugs is pretty likely to be missing (pip install pyHS100) . Depending on your installation there may be more.

Much of the application is configurable through the parameter.py file, but the default should give you an application that runs, including a default database. You probably will not be able to talk to your smartplugs, because the parameters do need to be told what and where (tcpip address) they are.

So to tell the application about your plugs. You should know the address of each plug. If not you might want to use an application like advanced ip scanner (google it). The smartplug gives up very little information on my scans, but run the scan with the plug plugged in and out, and the address that appears (or disappears) it the address of your device.

The section of parameters.py that locates the device looks like this:

```
       self.device_list       =  [
              { "name": "device_1",  "tcpip": "192.168.0.209", "more": None, "gui_label": None, "gui_combo": None  },
              { "name": "device_2",  "tcpip": "192.168.0.209", "more": None, "gui_label": None, "gui_combo": None  },
              { "name": "device_3",  "tcpip": "192.168.0.209", "more": None, "gui_label": None, "gui_combo": None  },
                                  ]
```

it is a list of dictionaries. The entry above is for 3 devices (but I only have one, so the tcpip address is repeated) For now only worry about the tcpip address and the name for the device; put in as many devices as you wish to control.

With the edited parameter file saved, restart the application. You should be able to "talk" to your devices. Not working? Double check or email me.

In the future I may try to add some autodetect features. There is a routine to discover the plugs in the pyHS100 library, but it does not work for me.

How To:... [edit]

- Edit the parameter file: Use the application button <Edit Parms> (after configuring for your editor) or any text editor suitable for Python (no tabs) on the file parameters.py. See also: The section below, Configuration Files For Python and SmartPlug GUI Images
- View the log: Use the application button (after configuring for your editor) or any text editor on the file smart_plug.py_log. This is the default file name, it can be changed in parameters.py.
- Create a new empty database: There is a button on the graphing application. First set the file name (including full path) to a non existent file.
- Backup the database: Just make a copy of the database. It is all in one file, but you can make multiple databases.
- Add a new device or change device data: edit the parameter file, device_list.
- Record data: Press the <Record On> button, to stop <Record Off> or close the application. See also: SmartPlug GUI Images
- Set a timer: See also: SmartPlug GUI Images
- See status: Of what, work in progress.
- Make a graph: Use the graphing application ... a whole section on this comming soon
- Export a csv file: See also: SmartPlug GUI Images
- Install: see section above.
- Debug: see section below
- Change Parameters: Same as edit the parameter file.
- Retrieve data from plug: On the GUI, but not implemented.
- Quick Restart of the application: As perhaps after a parameter change. Use the <Restart> button. See also: SmartPlug GUI Images

How to Use the Parameter File [edit]

You need a text editor sutiable for .py files to manage the parameter file (parameters.py) This includes most text editors. I particularity like:

```
    notepad++
    geany
```

You can also use the editor that comes in many python development environments, the simplest of which may be Idle. But there are many many others. If you are reading this you probably have some experience.

Once you configure an editor in parameters.py you can edit from the <Edit Parms> in the GUI (see below)

```
    When editing there are couple of gotchas to watch out for.
        * Python cares about capitalization, use the capitalization indicated in the default
          files and the example code.
        * Python also cares a lot about how lines are indented.  Do not change the indentation from
          the sample files, and always indent using spaces ( not tabs. most text editors will use spaces
          automatically for .py files, even if you use the tab key )
```

```
    First time editing of the parameters.py file
```

```
        Use your chosen text editor to open the file parameters
```

specify in parameters.py with

```
    self.ex_editor          =  r"D:\apps\Notepad++\notepad++.exe"   # use r" or the backslashes will not work, or you can use forward slashes instead they may be wrong
    but they work.
```

Working With Database Files [edit]

- Create a directory for your data and database -- application comes with defaults

- Run database definition routine button wf1

```
       You need to name a database, application will default
       You also need your sample file input, start with the one from
       the step above.
       look at output.......
```

The GUI [edit]

see: SmartPlug GUI Images

General [edit]

probably drop this see gui images

Debugging [edit]

There are several application outputs that may be useful for debugging.

- Watch the GUI
- Check the Python console.
- Look at the python log file (use the GUI button <Edit Log> (specify your editor in parameters.py first) or use your editor on the default name of the log file ...\smart_plug\smart_plug.py_log.

Most issues will probably be missing libraries, parameters.py issues, or just bugs in my code (email me).

Work Flow Overview [edit]

Just a bit of theory the may be helpful - Some Definitions [edit]

Databases

```
    These encode data in a structured and efficiently searchable format.  It also easily select subsets of
    data and can sort those results.  There are lots of different kinds, but one fairly standard one is a SQL or
    relational database.
```

Database File

```
    The file ( or one of the files ) where database information is kept.  We are using sqllite, which keeps a whole database
    in one file.  This makes it very easy to move and/or backup a database.
```

Table, Record, Column

```
    in a SQL data base data is stored in Tables ( many tables may be put in one database ).  A table consists of records ( also called rows. )  Each row is information about some "thing".  For example if the "thing" is a person a record
    might contain the person's first name, last name, date of birth.... The table is much like a spread sheet with
    the information on each person in a row.  Each of the items ( first name, last name, date of birth.... ) is called a column.
```

SQL - Structured Query Language

```
    This is the language used by relational databases.  Typically the system generated the required SQL and runs
    it.  The user interface often shows the SQL which is quite a bit easier to read than it is to write.
    It may give you useful feedback on what the system is doing.
```

Links [edit]

- Python Control of Smart Plugs main page for this project
- category page for this project

Categories: Python SmartPlug | Python