

Title: hardInfo.py developer documentation
Project: hardInfo
Host: GitHub
Date Started: March 18, 2022
Author: George Keith Watson

On 2022-03-18 I imported minimal set of GUI components from another project of mine, LinuxLogForensics, which is the code base of my fileHero project at GitHub, and started removing unneeded code from them to create a new project the purpose of which is to make Linux system hardware information available in a convenient form for Python developers.

Design Goals

The primary purpose of this software is to collect all of the hardware, firmware, and bios information possible using Linux commands, via subprocess.Popen(), and make it available in a structured set of Python classes. To see what is already available in the Python library for this purpose, see the output of the os.uname() function. Here's an example:

```
posix.uname_result(sysname='Linux', nodename='keithcollins-HP-ProBook-6450b',  
release='4.15.0-156-generic', version='#163-Ubuntu SMP Thu Aug 19 23:31:58  
UTC 2021', machine='x86_64')
```

The type of this output is <class 'posix.uname_result'>, which is a named tuple.

Here's what is produced on the same system with the Linux command: `uname -a`:

```
Linux keithcollins-HP-ProBook-6450b 4.15.0-156-generic #163-Ubuntu SMP Thu  
Aug 19 23:31:58 UTC 2021 x86_64 x86_64 x86_64 GNU/Linux
```

The named tuple type is immutable, so the built in Python function does have this advantage. However, the Python programmer should have a class representing this output as a set of properties so that methods can be added for any purpose, including security protocols like integrity checking against other system information. The named tuple can always be preserved in the vPython uname class as a property, and since it is presumably produced by a different source, a Python library function rather than the Linux command line via Python's subprocess.Popen class, it can itself be used for checking program correctness and system integrity.

os.uname() is an isolated example. The Python library does not have an equivalent of the Linux lshw command, for instance, which produces fairly comprehensive specification of installed hardware which the operating system builds at boot time. When run using sudo, mine produces about 1200 lines of output. I did an online search for "python lshw equivalent and found various third party libraries available for free, like those mentioned here:

<https://www.thepythoncode.com/article/get-hardware-system-information-python>

"How to Get Hardware and System Information in Python"

As a Python developer, it is handy to use third-party libraries that do the job you actually want, instead of reinventing the wheel each time. In this tutorial, you will be familiar with [psutil](#) which is a cross-platform library for process and system monitoring in Python, as well

as the built-in [platform](#) module to extract your system and hardware information in Python.

The Python platform module is tedious because you have to request one property at a time, so I wrote a method which puts them all into a map. I also wrote a general PropertySheet GUI component using tkinter.LabelFrame which displays the result.

The psutil project would be another alternative. It uses the same Linux command leveraging approach as this project does, but covers different commands and different subject matter. Here is the first paragraph of its PyPI Project description:

psutil (process and system utilities) is a cross-platform library for retrieving information on **running processes** and **system utilization** (CPU, memory, disks, network, sensors) in Python. It is useful mainly for **system monitoring, profiling and limiting process resources** and **management of running processes**. It implements many functionalities offered by classic UNIX command line tools such as *ps*, *top*, *iostat*, *lsof*, *netstat*, *ifconfig*, *free* and others.

API Design Goals

The API should have a consistency such that all objects, which initially are just JSON structures but which are now imported into classes, can be assembled into a single unified hardware and platform information structure. All information obtained will be store-able in timeStamped records in a table or tables so that history can be reported if changes are made. This also helps detect malware manipulating the operating systems files or data bases storing the hardware information. The OS collects the available hardware and other platform information on boot.

Goals:

There needs to be a CLI along with the API.

A GUI using a property sheet component and a treeview component will also be required.

The Console can be included along with tool configuration, saving, and piping.

Piping between tools can then be done using Python classes and data dictionary maps and JSON dict structures.

- Shell out to terminal for sudo.
- Write classes for components in lshw json / dict.
- Complete API on tools implemented so far.
- Run history can be used for tool memory.
- Save output to SQLite DB table.
- CLI for tools and admin implemented so far.
- GUI for tools and admin implemented so far.
- Build and test:
 - Makable source archive
 - Wheel
 - Debian executable
- API documentation for initial lshw implementation.
- Explain import integrity checking in programmer documentation.
- Plan CLI format, improvements over Linux versions where --json is available.
- Plan API capabilities, including those of hierarchical and relational databases.
- Possible application of text indexing.

- Plan GUI capabilities.
- Use of "less" for consistent formatting when piping input in, e.g. (see man pages):
\$: lscpu | less
- DB potential table potential for commands that have arguments that support the "list" argument, like lscpu.
- Need "comprehensive" getter methods
 - list all instances of a particular class of hardware, include all locations in the hierarchy of hardware where they occur.
 - find all instances of a particular class of hardware with a particular attribute name.
 - find all instances of a particular class of hardware with a particular attribute name which attribute has a particular value.
 - grep type searching of attributes:
 - by attribute name or value
 - by hardware instance type and attribute name or value,
 - ...

Separate, or sub project idea:

This produces a hierarchical database, so any search or filter method applicable to a hierarchical database should be considered.

- ◆ Attribute name indexes: Scan hierarchy for all occurrences of a particular attribute name, as if it were a column.
- ◆ Use hash table to record the attribute value as the key and the path to the location as the value, with a bucket for multiple instances of the same value.
- ◆ Search for exact match is then just a hash table access.
- ◆ Grep and fuzzy matching require a scan of the entire key set, so this is slower, $O(n)$.

List of Linux commands [possibly] to be included:

- uname
- lshw
- lscpu
- lsblk
- lsusb
- lspci
- lsscsi
- hdparm
- fdisk
- dmidecode
- free
- df, pydf
- dmesg
- biosdecode
- dig, host, ip, nmap, ping