

Re: Help With PyParsing of output from win32pdhutil.ShowAllProcesses()

## Re: Help With PyParsing of output from win32pdhutil.ShowAllProcesses()

---

*Source:* <http://coding.derkeiler.com/Archive/Python/comp.lang.python/2007-09/msg01451.html>

---

- *From:* Paul McGuire <[ptmcg@xxxxxxxxxxxxxxx](mailto:ptmcg@xxxxxxxxxxxxxxx)>
  - *Date:* Wed, 12 Sep 2007 01:21:08 -0700
- 

On Sep 11, 1:12 pm, Steve <[sreissc...@xxxxxxxxxx](mailto:sreissc...@xxxxxxxxxx)> wrote:

Hi All (especially Paul McGuire!)

Could you lend a hand in the grammar and paring of the output from the function win32pdhutil.ShowAllProcesses()?

This is the code that I have so far (it is very clumsy at the moment) :

<snip>

Many thanks!

Steve

Steve –

Well, your first issue is not a pyparsing one, but one of redirecting stdout. win32pdhutil.ShowAllProcesses does not *\*return\** the output you listed, it just prints it to stdout. The value returned is None, which is why you are having trouble parsing it (even after converting None to a string).

For you to parse out this data, you will need to redirect stdout to a string buffer, run ShowAllProcesses, and then put stdout back the way it was. Python's cStringIO module is perfect for this:

```
from cStringIO import StringIO
import sys
import win32pdhutil

save_stdout = sys.stdout
process_info = StringIO()
```

Re: Help With PyParsing of output from win32pdhutil.ShowAllProcesses()

## Re: Help With PyParsing of output from win32pdhutil.ShowAllProcesses()

```
sys.stdout = process_info

win32pdhutil.ShowAllProcesses()
sys.stdout = save_stdout
sProcess_Info = process_info.getvalue()
```

\*Now\* you have all that data captured into a processable string.

As others have mentioned, this data is pretty predictably formatted, so pyparsing may be more than you need. How about plain old split?

```
for line in sProcess_Info.splitlines()[1:]:
    data = line.split()
    print data
```

Done!

Still have an urge to parse with pyparsing? Here are some comments on your grammar:

– Your definition of `process_name` was not sufficient on my system. I had some processes running whose names includes numeric digits and other non-alphas. I needed to modify `process_name` to:

```
process_name = pyparsing.Word(pyparsing.alphanums+"_-")
```

– Similarly, some of my values returned by `ShowAllProcesses` had negative values, so your definition of `integer` needs to comprehend an optional leading '-' sign. (This actually sounds like a bug in `win32pdhutil` – I don't think any of these listed quantities should report a negative value.)

– Whenever I have integers in a grammar, I usually convert them to ints at parse time, using a parse action:

```
integer.setParseAction( lambda tokens : int(tokens[0]) )
```

– The tabular format of this data, and the fact that the initial entry in each row appears to be a label of some sort invites the use of the pyparsing `Dict` class. I note that you are already trying to extract keys from the parsed data, so it looks like you are already thinking along these lines. (Unfortunately, it is very likely you will get duplicate keys, since process names do not have to be unique – this will involve some loss of data in this example.) The `Dict` class auto-generates results names in the parsed results. `Dict` turns out to be awkward to use directly, so I added the `dictOf` method to simplify things. The concept of `dictOf(keyExpr,valueExpr)` is "parse a list of dict entries, each of which is a key-value pair; while parsing, label

Re: Help With PyParsing of output from win32pdhutil.ShowAllProcesses()

each entry with the parsed key." In your example, this would be:

```
ProcessList = heading + pyparsing.dictOf(process_name,  
pyparsing.OneOrMore(integer) )
```

The key is a leading process\_name, and the value is the following list of integers. With this, you can print out the results using:

```
data = ProcessList.parseString(sProcess_Info)
```

```
print "data keys:", data.keys()  
for k in sorted(data.keys()):  
print k, ":", data[k]
```

Getting:

```
BCMWLTRY : [684, 0, 0, 0, 54353920, 53010432]  
CLI : [248, 0, 0, 0, 171941888, 153014272]  
D4 : [2904, 0, 0, 0, 37527552, 36413440]  
F-StopW : [2064, 0, 0, 0, 33669120, 30121984]  
....
```

(again, note that the multiple entries for "CLI" have been reduced to a single dict entry)

You could get similar results using something like:

```
data = dict((vals[0],vals[1:]) for vals in  
map(str.split,sProcess_Info.splitlines()))
```

But then you would never have learned about dictOf!

Enjoy!  
-- Paul

.