Powered Device Analyzer
Application Note

Using PowerShell PDA with Python

Sifos Technologies

Introduction

Sifos Powered Device Analyzers are controlled by a software environment named PowerShell PDA. PowerShell PDA is furnished as a binary API, and includes a Tcl extension library. The available commands are documented in the PDA-602 Technical Reference Manual and PDA-604 Technical Reference Manual.

Tcl/Tk, a popular language in the realm of network testing, is available for Personal Computers running the Microsoft Windows Operating System or the Linux Operating System, as well as for proprietary Unix workstations (such as Sun). The degree of platform and operating system independence renders Tcl/Tk applications highly immune from updates in operating systems and computer hardware. At this time, PowerShell PDA is only supported on Microsoft Windows.

For those test developers who are working in Python, the **tkinter** package can be used to provide direct access to the complete set of PowerShell PDA Tcl extensions. The **tkinter** package is a “thin” object oriented binary package that is compiled against a specific version of Tcl/Tk.

Note that the **tkinter** package is used with Python version 3.0 and later, and is built against and installed with Tcl/Tk8.6. Earlier versions of Python use a package named **Tkinter** (note the upper case ‘T’), which is built against and installed with Tcl/Tk8.5.

Starting with version 1.17.0.5, PowerShell PDA supports 32-bit Tcl/Tk8.4, Tcl/Tk8.5, and Tcl/Tk8.6, and 64-bit Tcl/Tk8.6 on systems running Microsoft Windows.

Note that PowerShell PDA is also furnished with a binary API Library, furnished as a **.dll** format (Microsoft Windows). This binary API can be used with Python by way of the **ctypes** foreign function library. Examples of how to encapsulate the functions are provided with PowerShell PDA, but accessing PowerShell PDA using **tkinter** is the recommended approach.

The discussions below shows how to use **tkinter** in a text mode Python interpreter. Any of the examples as shown that cause output to ***stdout*** cannot be used in a Python IDLE application, which is a graphical interface, in which the ***stdout*** channel does not exist. While it is technically possible to redirect output from **tkinter** to a Tk widget, that is beyond the scope of this document.

Contents

Using PowerShell PDA with Python 3.x

Python version 3.7.4 was used for the following example statements. The tkinter package installed Tcl 8.6.9 ((32-bit or 64-bit, matching the Python installation).

Loading the PowerShell PDA Tcl Extension in Python 3.x

The PowerShell PDA Tcl extension is loaded by sourcing a runtime configuration file (pda600tclRC.tcl for Tcl, pda600wishRC.tcl for Tk) from a shell. The following Python statements show how to instantiate a Tcl interpreter and load the extension. This allows the various commands contained in the PowerShell PDA Tcl extension to be executed directly from a Python script:

```
import tkinter
tcl=tkinter.Tcl()
tcl.eval('puts $tcl_version')
rslt=tcl.eval('source "/Program Files (x86)/Sifos/PDA600/tcl/pda600tclRC.tcl"')
pdaver=tcl.eval('pda_version')
print(pdaver)
```

NOTE: this example will encounter an error if there are two or more PDA-600s connected to the host. If that situation exists, it is necessary to insert a statement to connect to a specific device before executing the **pda_version** command, for example:

```
tcl.eval('pda dev 1')
```

Executing PowerShell PDA Tcl Commands in Python 3.x using tcl.eval

Once the PowerShell PDA Tcl extension has been loaded in the Tcl interpreter which has been instantiated in the Python shell, any of the commands contained in that extension can be executed using **`tcl.eval`**. Here is an example code segment:

```
tcl.eval('pda_alt a;pda_polarity neg a')
tcl.eval('pda_source 50 slew fast on')
rslt=tcl.eval('pda_pdc stat')
print(rslt)
```

Once the PowerShell PDA extension has been loaded and the connection to the instrument established, the Python script appears to be correctly paced in terms of the **`tcl.eval`** method not returning until the Tcl command has completed execution.

NOTE: you must use the full PowerShell PDA Tcl command. For example, if you use the short form for the **polarity** command, you will encounter an error:

```
tcl.eval('pda_pol neg a')
```

The result will be:

```
>>> tcl.eval('pda_pol neg a')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
_tkinter.TclError: invalid command name "pda_pol"
```

Values that are displayed in the console with the Tcl 'puts' command will not be returned to the Python script by the `tcl.eval` method. Any value that is returned as a Tcl result (i.e. a 'return $somevar' command at the end of a proc) will be returned to the Python script by the `tcl.eval` method. For example, the call:

```
   rslt=tcl.eval('pda_pdc stat')
```

will cause the Python variable `rslt` to be set to Tcl result from the average power meter:

```
>>> rslt=tcl.eval('pda_pdc stat')
>>> print(rslt)
AVERAGE_POWER: 20.30 W
PEAK_POWER: 21.55 W
```

Python variables can be used in Tcl.eval calls. The Python variable needs to be processed with the **tkinter._stringify** method, and used in the statement passed as the argument to **tcl.eval** exactly as shown in the example below. In this example, the voltage level that the source is to be set to is contained in the Python variable ***vlts***:

```
vlts=tkinter._stringify("52.0")
tcl.eval('pda_source %(vlts)s on' %locals())
vport=tcl.eval('pda_vdc stat')
print(vport)
```

```
>>> vlts=tkinter._stringify("52.0")
>>> tcl.eval('pda_source %(vlts)s on' %locals())
''
>>> vport=tcl.eval('pda_vdc stat')
>>> print(vport)
VPORT: 52.1 VDC
```

Using PowerShell PDA with Python 2.x

Python version 2.7.6 (32-bit) was used for the following example statements. The Tkinter package installed Tcl 8.5.2.

Loading the PowerShell PDA Tcl Extension in Python 2.x

The PowerShell PDA Tcl extension is loaded by sourcing a runtime configuration file (pda600tclRC.tcl for Tcl, pda600wishRC.tcl for Tk) from a shell. The following Python statements show how to instantiate a Tcl interpreter and load the extension. This allows the various commands contained in the PowerShell PDA Tcl extension to be executed directly from a Python script:

```
import Tkinter
tcl=Tkinter.Tcl()
tcl.eval('puts $tcl_version')
tcl.eval('cd "/Program Files (x86)/Sifos/PDA600/bin"')
rslt=tcl.eval('source "/Program Files (x86)/Sifos/PDA600/tcl/pda600tclRC.tcl"')
pdaver=tcl.eval('pda_version')
print(pdaver)
```

NOTE: this example will encounter an error if there are two or more PDA-600s connected to the host. If that situation exists, it is necessary to insert a statement to connect to a specific device before executing the **pda_version** command, for example:

```
tcl.eval('pda dev 1')
```

Executing PowerShell PDA Tcl Commands in Python 2.x using tcl.eval

Once the PowerShell PDA Tcl extension has been loaded in the Tcl interpreter which has been instantiated in the Python shell, any of the commands contained in that extension can be executed using **`tcl.eval`**. Here is an example code segment:

```
tcl.eval('pda_alt a;pda_polarity neg a')
tcl.eval('pda_source 50 slew fast on')
rslt=tcl.eval('pda_pdc stat')
print(rslt)
```

Once the PowerShell PDA extension has been loaded and the connection to the instrument established, the Python script appears to be correctly paced in terms of the **`tcl.eval`** method not returning until the Tcl command has completed execution.

NOTE: you must use the full PowerShell PDA Tcl command. For example, if you use the short form for the **polarity** command, you will encounter an error:

```
tcl.eval('pda_pol neg a')
```

The result will be:

```
-------------------------------------------------------------------------
TclError                                Traceback (most recent call last)
<ipaython-input-31-0a08d1e4f346> in <module>()
----> 1 tcl.eval(' pda_pol neg a ')

TclError: invalid command name " pda_pol "
```

Values that are displayed in the console with the Tcl 'puts' command will not be returned to the Python script by the `tcl.eval` method. Any value that is returned as a Tcl result (i.e. a 'return $somevar' command at the end of a proc) will be returned to the Python script by the `tcl.eval` method. For example, the call:

```
   rslt=tcl.eval('pda_pdc stat')
```

will cause the Python variable `rslt` to be set to Tcl result from the average power meter:

```
@TestBed64[c:johns]|10> rslt=tcl.eval('pda_pdc stat')
@TestBed64[c:johns]|11> print(rslt)
AVERAGE_POWER: 20.30 W
PEAK_POWER: 21.55 W
```

Python variables can be used in Tcl.eval calls. The Python variable needs to be processed with the **tkinter _stringify** method, and used in the statement passed as the argument to **tcl.eval** exactly as shown in the example below. In this example, the voltage level that the source is to be set to is contained in the Python variable ***`vlts`***:

```
vlts=Tkinter._stringify("52.0")
tcl.eval('pda_source %(vlts)s on' %locals())
vport=tcl.eval('pda_vdc stat')
print(vport)

@TestBed64[c:johns]|14> vlts=tkinter._stringify("52.0")
@TestBed64[c:johns]|15> tcl.eval('pda_source %(vlts)s on' %locals())
@TestBed64[c:johns]|16> vport=tcl.eval('pda_vdc stat')
@TestBed64[c:johns]|17> print(vport)
VPORT: 52.1 VDC
```