Using the Vaunix interface DLLs from TcL under Windows

TcL provides a rich scripting language, and is designed to be extensible. Two approaches are described below. The first uses the traditional TcL C extension template, where an interface wrapper is written in C that manages the translation of arguments and return values from the DLL. The second uses the TcL dll caller extension. Both descriptions are based on published documentation, and have not been tested. Developers are encouraged to share feedback regarding their results using these approaches.

Writing a TcL extension

The traditional way of extending TcL is by writing an extension in C or another suitable language which encapsulates the functions of a DLL and handles the argument and return value translations. This kind of TcL extension can and should also provide error checking to simplify debugging of TcL applications that use the extension.

An example of a TcL extension is provided at http://core.tcl.tk/sampleextension/info/ee54bac585

A tutorial level description of a basic TcL extension is provided at http://wiki.tcl.tk/11153

A tutorial level description of how to build a basic TcL extension is provided at http://wiki.tcl.tk/28541

Using the TcL DLL caller extension

The TcL extension, "yet another DLL caller" is described at http://wiki.tcl.tk/12264

The package, named "DLL" has a number of commands that can be used to directly access the Lab Brick DLLs from TcL. Basically the process is:

1) Install the DLL extension into TcL

2) Use it to load the Lab Brick DLL. You must use the version of the Lab Brick DLL which has the stdcall calling convention. Assign a namespace to the DLL, such as atn for the attenuator:

```
::dll::load VNX_atten -> atn
```

3) Assign the functions you want to use from the Lab Brick DLL to new TcL commands within the atn namespace:

```
::atn::cmd "int fnLDA_GetNumDevices()"
::atn::cmd "void fnLDA_SetTestMode(int)"
::atn::cmd "int fnLDA_InitDevice(int)"
::atn::cmd "int fnLDA_SetAttenuation(int, int)"
::atn::cmd "int fnLDA_CloseDevice(int)"
```

Vaunix Technology Corporation
242 Neck Rd
Haverhill, MA 01835
USA
+1 978-662-7839

4) Call the Lab Brick DLL function to get the number of devices, for an attenuator this is fnLDA_GetNumDevices using the newly defined commands:

```
set w1 [::atn::fnLDA_GetNumDevice]
```

5) If the device count in w1 is non-zero, then a device is attached, and you can use the number of the device (1,2,3 etc.) as the device ID for all Lab Brick DLL functions that require a DeviceID. Note that we have skipped the step of obtaining the actual mapping between deviceID numbers and devices, so this approach will only work in simple configurations where the Lab Bricks are connected when the TcL script runs and their connections do not change while the script is operating. Adding or removing Lab Bricks will invalidate the mapping between deviceID values and hardware devices.

6) Use the deviceID values, 1 for the first attenuator, 2 for the second, etc. to call whatever other functions are desired. Start with the InitDevice function, and end with the close function.

```
set deviceID 1
set attenuate 40
set w2 [::atn::fnLDA_InitDevice $deviceID]
set w3 [::atn::fnLDA_SetAttenuation $deviceID $attenuate]
set w4 [::atn::fnLDA_CloseDevice $deviceID]
```

7) It appears to be possible to use the DLL caller to obtain the array of deviceID values from the Lab Brick DLL if necessary, and then manage the deviceIDs from the returned array of data, but the complexity of doing so may be avoided for simple hardware configurations that do not change during usage.

Vaunix Technology Corporation
242 Neck Rd
Haverhill, MA 01835
USA
+1 978-662-7839