KTL Keywords HowTo

Installing kroot

First, make sure a kroot directory is installed. Keck uses the SVN version control system, and protects their repository behind an account system. Unlike Github, where commits are protected but anyone can check out, svn/kroot can only be checked out if you have a Keck account.

I don't know how to go about checking out the kroot source code repository, since I don't have a Keck account. Once someone checks out the source code, it should be under ~/svn/kroot, aka, /home/<usrname>/svn/kroot. Navigate to the source code directory ~/svn/kroot, and run 'make install'. This should make a complete kroot install, in the directory /opt/kroot. (Note: I'm not sure if the bash shell variables need to be defined first.)

Setting up bash shell variables

These are commands to set up standard Lick Observatory path variables:

```
export KROOT=/opt/kroot
export RELDIR=$KROOT/rel/default
export LROOT=/usr/local/lick
```

You can add that exact text to your .bashrc file, to have those variables established when you start a new shell. Handy tip, run 'exec bash' to restart your current bash session.

To get the commands 'gshow' and 'gwrite' known when using cassini, I also had to add /opt/kroot/bin to my PATH. Using bash:

```
export PATH=/opt/kroot/bin:$PATH
```

If you're using a different shell, the syntax might differ from the above commands.

KTL Keywords Overview

The Keck Task Library (KTL) is a way of providing a unified, consistent interface with all the various machines used at the Keck and Lick Observatories. The basic architecture is a KTL keyword server, which takes keyword: value pairs and translates them into an interface for the device being controlled. A client program accesses the server, requesting and sending ktl keyword: value pairs.

Multiple clients can access a single ktl keyword server. As an example, shanegcam is a ktl keyword server that controls the Shane guider camera, and runs on the machine Cassini on Mt Hamilton. The guider GUI can access shanegcam, as well as a bash session on Cassini. Also, with the correct firewall ports opened, the machine Noise can access shanegcam remotely from the CCD lab at UCSC. These three different clients can all access the exact same server, at the same time.

Using the bash shell with KTL keywords
The typical commands are ‘gshow’ and ‘modify’. These shell commands read and write KTL keywords, respectively. To start, enter ‘gshow -h’. This brings up the help overview for the KTL keyword shell interface. Since there are a great many options, the help interface is divided up into several different topics.
List those topics with ‘gshow -h topics’, and display a particular topic with ‘gshow -ht <topicname>’.

Quick start using bash

- ‘gshow -h’ displays the help overview for the KTL keyword bash interface.
    - Since there are a great many options, the help interface is divided up into several different topics.
    - ‘gshow -h topics’ lists available help topics
    - ‘gshow -ht <topicname>’ displays a particular help topic
- ‘gshow’ lists all available ktl keyword services.
- ‘gshow -s <servicename>’ displays all keyword:value pairs
- ‘gshow -s <servicename> <keyword>’ displays a particular keyword:value pair
- ‘gshow -s <servicename> keywords’ displays all keywords, with associated metadata.
    - Metadata shows a brief description of the keyword, whether it’s writable or read-only, data type, and expected format.
    - c2 indicates a writable keyword, c- indicates a read-only keyword
- ‘modify -s <servicename> <keyword>=<value>’ writes a new value to the keyword

Examples

Invoking ‘gshow -s <servicename>’ prints all available keywords for that service:

```
[user@groundstrap ~]$ gshow -s shanegcam
ACCTIME =  2.594 s
ACQCOUNT =  15542
ACQPHASE = done
BINNING = 1,1
CAMSTATUS =  20073
(and so on...)
```

Display a certain keyword with ‘gshow -s <servicename> <keyword>’:

```
[user@groundstrap ~]$ gshow -s shanegcam currtemp
CURRTEMP = -60.11
```

Modify a keyword with ‘modify -s <servicename> <keyword>=<value>’:

```
[user@groundstrap ~]$ modify -s shanegcam cooltarg=-60
setting cooltarg = -60 (wait)
```

Keywords are case insensitive, e.g., asking for ‘currtemp’ is the same as asking for ‘CURRTEMP’ is the same as ‘currteMP’.
Service names **are** case sensitive, e.g. ‘shanegcam’ is not the same as ‘Shanegcam’ or ‘SHANEGCAM’.

KTL keywords are capable of holding arrays and array-likes. E.g., a crop frame keyword would look like this:

```
WINDOW =
   hbeg:    140
   hend:    820
   vbeg:    192
   vend:    820
```

While the above looks like a dictionary, it is modified by passing an array of numbers. See the section on Pie for an example.

Using ktl python
Correctly importing the ktl python module into a python environment involves some complicated pathing. This pathing is handled by a SPG homebrew python interpreter called ‘kpython’, which is installed as part of kroot. Kpython acts as either a macro preprocessor that runs when ‘make install’ is executed, or can be used to start an interactive session just like regular python, by executing ‘kpython’ or ‘kpython3’ in the shell. Beyond that, I don’t understand how it is supposed to work.

When not using kpython, the paths must be set up manually. The ktl keyword source code provides documentation on how to manage the paths when not using kpython, **however what the documentation says does not work.** Instead, for a system with a typical setup, add the needed paths after starting a python session by using the following commands:

```
        import sys

        sys.path.extend(['/opt/kroot/rel/default/lib/python',
                '/opt/kroot/rel/default/lib',
                '/usr/local/lick/lib/python',
                '/usr/local/lick/lib'])
```

These commands make the installed kroot and lroot directory accessible from your python session. (Notes: (1) DO NOT try to import from the source code directory ~/svn/kroot. It will not work, as this will ignore all the macro substitutions that happen when ‘make install’ is executed; (2) `/opt/kroot` is typical install point at UCO, but at Keck the conventional install point is `/opt`) From here, you should be able to import the ktl module.

To see how to use the ktl module, refer to the documentation included in the ktl python source code. A good place to start is code introspection using the ‘help()’ function; this displays all the docstrings associated with the object/module being examined. E.g., entering into the python shell ‘help(ktl)’ will display the documentation for all the classes and functions in ktl. To examine something specific, like the Service class, enter ‘help(ktl.Service)’.

To examine the source code directly, the source code directory for ktl python is ~/svn/kroot/ktl/keyword/python. A collected version of the documentation, along with examples, is in the sphinx subdirectory, ~/svn/kroot/ktl/keyword/python/sphinx.

Notes on keyword formats

- Ktl keywords that take array values can be written to with lists.
- Enums are handled by either passing the enum name as a string, or passing the enum value.
- Values can be passed as ints, floats, etc, or as string equivalents. Eg, both 12.3 and ‘12.3’ are valid values to pass to a keyword.

Pie: spinning up a dummy service

The ktl keyword service ‘pie’ is a dummy service, meant for testing code. To spin it up:

- Execute ‘traffic’ in bash shell. Traffic handles backend communications in the ktl service, it only needs to be started once, at boot.
- Run ‘$RELDIR/sbin/pied’
    - This only runs while the terminal is active. Upon closing the terminal, pie stops
    - I’m pretty sure this isn’t the ‘correct’ way to do this, but run ‘$RELDIR/sbin/pied &’ to have it run in the background. No need to ‘disown’, pie will continue to run after closing the terminal

Pie provides a large number of examples of ktl keywords, and is useful for testing if the data format is being handled correctly by a program.

For example, dealing with ktl keywords that hold arrays:

Using the bash shell:

```
[user@groundstrap ~]$ gshow -s pie integer_array
INTEGER_ARRAY =
   Element_0:    0
   Element_1:    0
   Element_2:    0
   Element_3:    0
   Element_4:    0
   Element_5:    0
   Element_6:    0
   Element_7:    0
   Element_8:    0
```

```
    Element_9:    0
[user@groundstrap ~]$ modify -s pie integer_array='0 1 2 3 4 5 6 7 8 9'
setting integer_array =
    Element_0:    0
    Element_1:    1
    Element_2:    2
    Element_3:    3
    Element_4:    4
    Element_5:    5
    Element_6:    6
    Element_7:    7
    Element_8:    8
    Element_9:    9 (wait)
[user@groundstrap ~]$ gshow -s pie integer_array
INTEGER_ARRAY =
    Element_0:    0
    Element_1:    1
    Element_2:    2
    Element_3:    3
    Element_4:    4
    Element_5:    5
    Element_6:    6
    Element_7:    7
    Element_8:    8
    Element_9:    9
```

Using ktl python:

```
[user@groundstrap ~]$ kpython3
Python 3.6.8 (default, Apr  2 2020, 13:34:55)
[GCC 4.8.5 20150623 (Red Hat 4.8.5-39)] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import ktl
>>>
>>>
>>> pie = ktl.Service('pie')
>>>
>>> integer_array = pie['INTEGER_ARRAY']
>>>
>>> print(integer_array.read())
```

```
    Element_0:    1
    Element_1:    0
    Element_2:    0
    Element_3:    0
    Element_4:    0
    Element_5:    0
    Element_6:    0
    Element_7:    0
    Element_8:    0
    Element_9:    0
>>>
>>> technically_a_list = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>>
>>> integer_array.write(technically_a_list)
4
>>>
>>> print(integer_array.read())

    Element_0:    0
    Element_1:    1
    Element_2:    2
    Element_3:    3
    Element_4:    4
    Element_5:    5
    Element_6:    6
    Element_7:    7
    Element_8:    8
    Element_9:    9
>>>
```