Tcl/Tk/KTL Interface

File: 3.5.6 (KSDs), Library KSD Binder, Summit KSD Binder

Circulation: Software Coordination Committee, Software Group

Table of Contents

1.0 Introduction

This document describes a way of writing Tcl/Tk [*Tcl/Tk reference*] applications that use KTL [*KTL reference*] to interact with underlying subsystems. A new `ktl` command has been added to the standard set of Tcl commands, and corresponding `ktcl` (a non-windowing Tcl shell, c.f. `tclsh`) and `ktk` (a windowing Tcl shell, c.f. `wish`) applications have been written. `ktk` has been used to implement the new Keck II DCS GUI.

The current versions of `ktcl` and `ktk` are called `ktcl7.5` and `ktk4.1` and are layered on top of Tcl v7.5 and Tk v4.1. By convention, `ktcl` and `ktk` soft links reference the most recent versions. As new versions of Tcl and Tk are released, the `ktcl` and `ktk` names will be changed accordingly and the soft links will be updated.

The next section gives a tutorial introduction to the use of `ktcl` and `ktk`. Subsequent sections provide more detailed information on installation and usage.

2.0 Tutorial

This section assumes that `ktcl` and `ktk` are installed and in your binary search path (refer to Section 3.0 for installation details). It also assumes that you have a basic familiarity with Tcl and Tk.

2.1 `ktcl` tutorial

Everything that's in `ktcl` is also in `ktk` (apart from command editing). The `ktcl` tutorial illustrates how to read and write KTL keywords. The `ktk` tutorial follows up with how to monitor keywords and interface with Tk widgets.

2.1.1 Starting up

Type `ktcl`. This gives a `ktcl7.5%` (the `7.5` is the Tcl version number) prompt.

```
prompt% ktcl
ktcl7.5%
```

You now have access to all the commands offered by the standard `tclsh` shell, and also to a `ktl` command, which offers access to KTL keyword libraries.

2.1.2 Reading a keyword

The following commands read the Keck II DCS UTC keyword.

```
ktcl7.5% ktl open dcs2
dcs2
```

Call ktl_open() to open a connection to the KTL dcs2 keyword library.

```
ktcl7.5% ktl link dcs2 utc
utc
```

Associate dcs2's utc keyword with the global Tcl variable utc.

```
ktcl7.5% ktl read utc
 19:58:13.46
```

Read the current value of the keyword, formatting the value as a user-oriented ASCII string.

```
ktcl7.5% ktl read utc -binary
 8.47829e+08
```

Read the current value of the keyword, returning its binary value (here, seconds since 01-Jan-1970).

This can easily be packaged into a small `show` procedure.

```
ktcl7.5% proc show {minuss service keyword} {
ktcl7.5>     ktl open $service
ktcl7.5>     ktl link $service $keyword
ktcl7.5>     ktl read $keyword
ktcl7.5> }
```

The procedure is defined. The "minuss" argument is ignored and is merely to support existing syntax.

```
ktcl7.5% show -s dcs2 utc
 20:10:42.67
```

The procedure is invoked.

```
ktcl7.5% puts $utc
 20:10:42.67
```

The global variable contains the most recent value.

2.1.3 Writing a keyword

Writing a keyword is analogous to reading a keyword. The following commands select a Keck II DCS pointing origin.

```
ktcl7.5% ktl open dcs2
dcs2
```

Call ktl_open() to open a connection to the KTL dcs2 keyword library.

```
ktcl7.5% ktl link dcs2 poname
poname
ktcl7.5% ktl link dcs2 poselect
poselect
```

Associate dcs2's poname and poselect keywords with global Tcl variables of the same name.

```
ktcl7.5% ktl write poname REF
REF
ktcl7.5% ktl write poselect 1
1
```

Write the keywords.

This can easily be packaged into a small `modify` procedure.

```
ktcl7.5% proc modify {minuss service keyval} {
ktcl7.5>     ktl open $service
ktcl7.5>     ktl set list [split $keyval =]
ktcl7.5>     set keyword [lindex $list 0]
ktcl7.5>     set value [lindex $list 1]
ktcl7.5>     ktl link $service $keyword
ktcl7.5>     ktl write $keyword $value
ktcl7.5> }


ktcl7.5% modify -s dcs0 poname=ref
ref
ktcl7.5% modify -s dcs0 poselect=1
1
```

The procedure is defined. The "keyval" argument is of the form "keyword=value".

The procedure is invoked.

2.1.4 Returning information about a keyword

The following `keyinfo` procedure reports a keyword's aliases, type and units.

```
ktcl7.5% proc keyinfo {minuss service keyword} {
ktcl7.5>     ktl open $service
ktcl7.5>     ktl link $service $keyword
ktcl7.5>     puts "info about $service $keyword keyword"
ktcl7.5>     foreach info {aliases type units} {
ktcl7.5>         puts "$info = [ktl ioctl $info $keyword]"
ktcl7.5>     }
ktcl7.5> }


ktcl7.5% keyinfo -s dcs2 poname
info about dcs2 poname keyword
aliases = pointing-origin-name
type = string
units = {}
```

The procedure is defined.

The procedure is invoked.

This is the only information that can currently be returned.

2.1.5 Command editing

`ktcl` is linked with the GNU `readline` library which means that all the "normal" command editing keys work, including all the arrow keys (which is all that most people will use) and many EMACS editing keys (I don't know many of these, but do use Ctrl-A to go to the beginning of the line and Ctrl-E to go to the end of the line).

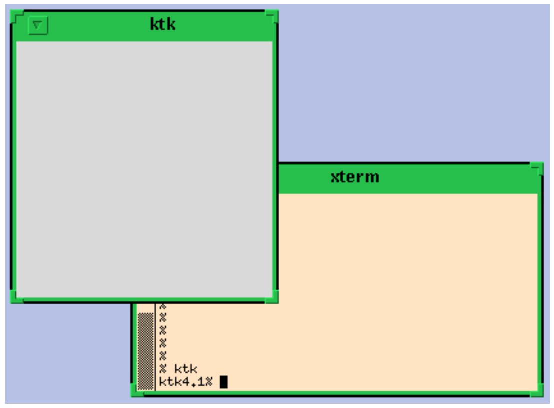
Tcl in any case, like most shells, supports ";" as a command separator character, so multiple commands can be written on a single line and then recalled and edited using the arrow keys.

2.2 `ktk` tutorial

You should have at least glanced at the ktcl tutorial, which illustrates basic KTL interactions, before trying out this tutorial. The `ktk` tutorial explains how to monitor keywords and interface with Tk widgets.

2.2.1 Starting up

Type `ktk`. This gives a square `ktk` window and a `ktk4.1%` (the `4.1` is the Tk version number) prompt.



You now have access to all the commands offered by the standard `tclsh` shell, and also to a `ktl` command, which offers access to KTL keyword libraries.

2.2.2 Monitoring a keyword

The following illustrates how to monitor the Keck II DCS UTC keyword.

```
ktk4.1% ktl link dcs2 utc
utc


ktk4.1% ktl monitor utc
utc


ktk4.1% puts $utc
00:03:14.75
ktk4.1% puts $utc
00:03:18.25
```

`ktl link` does an automatic `ktl open` if need be.

Whenever a monitor is delivered, the global `utc` variable will be updated.

It can be seen that the variable is changing autonomously.

A script can optionally be specified.

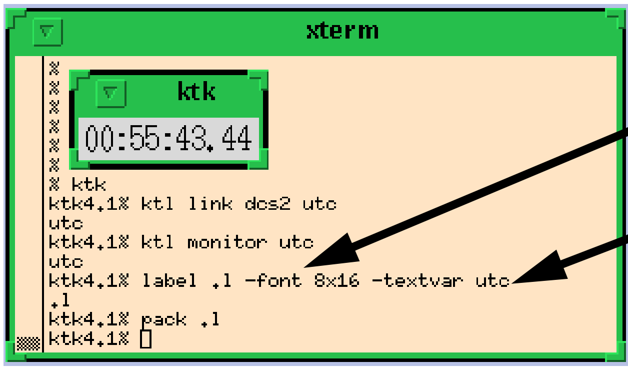
```
ktk4.1% ktl link dcs2 utc
utc
ktk4.1% ktl monitor utc {
ktk4.1>     puts “utc = $utc”
ktk4.1> }
utc
ktk4.1%
utc = 00:39:08.70
utc = 00:39:08.75
utc = 00:39:09.25
utc = 00:39:09.75
utc = 00:39:10.25
utc = 00:39:10.75
.
.
```

Whenever a monitor is delivered, the global `utc` variable is updated and then the supplied script is executed.

It can be seen that, after an initial “pump-priming” monitor, the variable is changing at 2Hz.

2.2.3 Updating the display

This is really nothing to do with KTL, but it demonstrates how easy is it to do!



This is a fixed width font (it looks better).

The label will automatically display whatever is in the `utc` global variable.

3.0 Installation

ktcl7.5 and ktk4.1 are dependent on the following:

1. Tcl v7.5 (assumed installed according to distribution defaults, except that the include file is assumed renamed to `tcl7.5.h`)
2. Tk v4.1 (assumed installed according to distribution defaults, except that the include file is assumed renamed to `tk4.1.h` and edited to reference `tcl7.5.h`)
3. GNU readline (needed by `ktcl` only)
4. MUSIC shareable library (needed if will be used with any keyword libraries that require it)
5. KTL (any version)
6. `/kroot/etc` v8.4 (mods to support SunOS and Solaris builds from the same makefile)

To install it, unpack it into `$KROOT/ktl/keyword/tcl` and type `make`. If all the above are correctly installed, it should build without problem under SunOS and Solaris. Of course, life is never that easy.

If you want the `ktcl` and `ktk` commands to reference the current versions, make appropriate links in `$RELDIR`.

4.0 Bibliography

1. Tcl/Tk reference
2. KTL reference

5.0 Changes to this document

- Version 1, the first version, was released in November 1996. It was written by William Lupton and described ktcl/ktk v1.7 (same as v1.6 but with minor bug fixes made as the manual was being written!).

Appendix A ktl - built-in command providing access to KTL keyword libraries

A.1 Synopsis

`ktl` *`option name ?arg arg ...?`*

A.2 Description

This command provides access to KTL keyword libraries. *`option`* identifies the operation (many of the options have the same names as user-callable KTL routines). Depending on *`option`*, *`name`* is either a service or a keyword name.

Valid options, which cannot currently be abbreviated, are:

TABLE 1. Options to the `ktl` command

Name	Description
`open`	open a connection to a KTL service (c.f. `ktl_open()`)
`link`	link a (service,keyword) pair, giving it a name used in subsequent operations
`monitor`	enable a monitor on keyword change, optionally executing a Tcl script (c.f. `ktl_read(KTL_CONTINUOUS)`) (`ktk` only)
`read`	read the value of a keyword (c.f. `ktl_read()`)
`write`	write the value of a keyword (c.f. `ktl_write()`)
`ioctl`	return information about a keyword

The obvious `close`, `unlink` and `unmonitor` have not yet been implemented.

A.2.1 `ktl open` *`service`*

Open a connection to a KTL service by calling `ktl_open()`. It returns the service name as its value, which should be used in subsequent `ktl link` commands.

If this service has already been opened, it returns immediately without doing anything. Thus the same command can be executed many times without mishap.

The command is optional because `ktl link` will always do an automatic `ktl open` if necessary.

For example:

```
ktk4.1% ktl open hires
hires
```

A.2.2 `ktl link` *`service keyword ?varname?`*

Link a (service,keyword) pair, associating it with a global Tcl variable. It returns the variable name as its value, which should be used in subsequent `ktl monitor`, `read`, `write` and `ioctl` operations.

The name assigned to the (service,keyword) pair is by default the keyword name. However, the name can be overridden by an optional extra argument. This is sometimes useful when the keyword name would cause a conflict with some other global variable name (e.g. the same keyword name exists in several keyword libraries).

If the service has not already been opened, an automatic `ktl open` is performed.

If the global variable already exists, it is not touched. Otherwise, it is created and is given an initial value of `<udf>` (for "undefined"). This behavior is not currently controllable.

For example:

```
ktk4.1% ktl link dcs2 axestat
axestat

ktk4.1% ktl link dcs2 axestat axestatVar
axestatVar
```

A.2.3 `ktl monitor` *`varname ?script?`* `?-binary?`

This option is only available in `ktk`.

Monitor the specified keyword, optionally executing a script whenever a monitor is triggered. The variable name is returned as the command value.

`ktl link` must already have been used to link the service and keyword. Usually, *`varname`* is the keyword name, but `ktl link` may have overridden this.

When a monitor is triggered, the *`varname`* global variable is updated with the new keyword value. By default this is the ASCII (external) representation of the value (usually this is what GUIs will want to use), but `-binary` will force use of the binary (internal) representation.

If a script is specified, it can be any set of Tcl commands and will be invoked after the global variable has been set.

The `KTL_NOPRIME` flag is always used when enabling monitors. This means that no explicit read of the keyword to obtain an initial value will take place. Some keyword libraries will handle this internally. Others will not.

Numeric values which do not begin with a sign will have a leading space inserted. This is to improve visual effect (a column of left-justified numbers, some of which have signs and some of which don't looks horrible!). This behavior cannot currently be controlled.

For example, using ASCII representation (degrees in this case):

```
ktk4.1% ktl link dcs2 az
az
ktk4.1% ktl monitor az {puts $az}
az
ktk4.1%
 354.8103
 354.8116
 354.8130
  .
```

.

...and using binary representation (radians in this case):

```
ktk4.1% ktl link dcs2 az
az
ktk4.1% ktl monitor az {puts $az} -binary
az
ktk4.1%
 6.22573
 6.22576
 6.22578
```

A.2.4 `ktl read` *`varname`* `?-binary?`

Read the specified keyword, setting the associated global variable and returning the keyword value as the command value.

`ktl link` must already have been used to link the service and keyword. Usually, *`varname`* is the keyword name, but `ktl link` may have overridden this.

By default the ASCII (external) representation of the value (usually this is what GUIs will want to use) is used, but `-binary` will force use of the binary (internal) representation.

Numeric values which do not begin with a sign will have a leading space inserted. This is to improve visual effect (a column of left-justified numbers, some of which have signs and some of which don't looks horrible!). This behavior cannot currently be controlled.

The read always blocks on completion (i.e. the only flag currently supported is `KTL_WAIT`). Time-out intervals are keyword library-specific and are not controllable at this level.

For example:

```
ktk4.1% ktl link dcs2 instrume
instrume
ktk4.1% puts "instrument is [ktl read instrume]"
instrument is LRIS
```

A.2.5 `ktl write` *`varname value`* `?-binary?`

Set the specified keyword to the specified value, returning the value as the command's value.

`ktl link` must already have been used to link the service and keyword. Usually, *`varname`* is the keyword name, but `ktl link` may have overridden this.

By default the value is assumed to be the ASCII (external) representation (usually this is what GUIs will want to use), but `-binary` will force use of the binary (internal) representation.

The write always blocks on completion (i.e. the only flag currently supported is `KTL_WAIT`). Time-out intervals are keyword library-specific and are not controllable at this level.

For example:

```
ktk4.1% ktl link dcs2 telfocus
telfocus
ktk4.1% ktl link dcs2 secmove
```

```
secmove
ktk4.1% ktl write telfocus -5
-5
ktk4.1% ktl write secmove 1
1
```

A.2.6 `ktl ioctl` *`option varname`*

Return information about the specified keyword (the information is returned as the command value).

`ktl link` must already have been used to link the service and keyword. Usually, *`varname`* is the keyword name, but `ktl link` may have overridden this.

The currently supported options (a subset of those supported at the KTL level) are:

TABLE 2. Options to the `ktl ioctl` command

Name	Description
`aliases`	return a list of the keyword's alias names
`type`	return the keyword's data type, currently one of the strings `double`, `float`, `int`, `bool`, `string` and `<unk>` (for "unknown")
`units`	return a list of the keyword's units strings

Returned lists are not enclosed in `{}` characters if they have only a single element. Use Tcl list-handling commands to operate on them.

For example:

```
ktk4.1% ktl link dcs2 secmove
secmove
ktk4.1% ktl ioctl aliases secmove
move-secondary
ktk4.1% ktl ioctl type secmove
bool
ktk4.1% ktl ioctl units secmove
no,yes
```

(the latter shows a deficiency in the cake library; the units strings should have been returned as `{no yes}`)

A.3 Error handling

All KTL errors are detected and handled according to normal Tcl conventions: the command fails and returns an error string as its value. In a Tk environment, this puts up a nice dialog box and it's usually pretty easy to see what the problem is.

For example:

```
ktk4.1% ktl open nonex
ktl_load_open.c, 200: ld.so.1: ktk: fatal: \
      /home/wlupton/kroot/rel/default/lib/libnonex_keyword.so.0.0: \
      can't open file: errno=2: /home/wlupton/kroot/rel/default/lib/ \
      libnonex_keyword.so.0.0.
ktl_open.c, 401: can't open shareable library: nonex_keyword.
```

```
can't open shareable library: nonex_keyword.

ktk4.1% ktl link dcs2 badkey
Invalid keyword name: badkey
```

As yet there is no re-direction of low-level error messages to a log file. All such messages are printed to standard error (this is the KTL default).

A.4 Bugs

Only actual known bugs are listed here. New features are listed in Appendix C.

1. Events can be lost by keyword libraries which do not support the `KTL_FDREG` ioctl and in which files can asynchronously be opened or closed (e.g. those that use RPCs).
2. Only one monitor can be enabled per keyword (this is actually a KTL restriction and could be worked around at the cost of increased complexity).

Appendix B Other commands

This section contains reference documentation on other commands (which did not warrant a private appendix each).

B.1 ktcl - version of tclsh with built-in KTL support

TBD. Refer to Section 2.1 on page 2 for tutorial information.

`ktcl` will probably be removed in the near future since it is now possible to load Tcl packages dynamically into `tclsh`. If this is done, command editing features will be lost. This is not too serious a problem because most applications will use `ktk` (or its replacement) and there are Tk consoles available which offer equivalent editing features.

B.2 ktk - version of wish with built-in KTL support

TBD. Refer to Section 2.2 on page 4 for tutorial information.

ktk will probably be removed in the near future since it is now possible to load Tcl packages dynamically into `wish`.

B.3 xhandler - enable an X error handler

B.3.1 Synopsis

`xhandler`

B.3.2 Description

Enable an X error handler using the standard Tk facilities. Whenever an X error occurs, it will be quietly ignored.

This command was added when Tk v3.6 was still being used. This had a bug in it which could crash long-running applications. No crashes have been seen with the Tk v4.1 version.

Once an X error handler has been enabled, it cannot currently be removed.

Appendix C Future developments

This is a wish-list of future developments. They are roughly prioritized. See also the README file in the source directory.

1. Convert to use dynamic package loading
2. Support `close`, `unlink` and `unmonitor` options
3. Support more ioctls and try and do so in a generic way
4. Add commands to return statistics (play game of returning information if trailing parameters are omitted; c.f. set)
5. Provide option of having no linked global variable; explore use of associative arrays rather than polluting the name-space with lots of globals (OK for small applications...)
6. Allow the KTL service to default to the last referenced service; will allow `ktl link` to be optional under most circumstances (could also support `-service` option on `ktl read` etc.)
7. Provide error logging control
8. Fix parsing of options (yukky now)
9. Provide run-time debug control
10. Support automatic trailing arguments to monitor scripts (e.g. service name, keyword name, error information), controllable via an extra option for upwards compatibility
11. Support Tcl lists for variables and values so can operate on several at a time
12. Support read/write flags other than `KTL_WAIT`; *could* implement command completion at the Tcl level if had to but better at KTL level (proceed on KTL support anyway; will need to keep "active list" at KTL level)
13. Work around KTL bug that prevents having more than one monitor per variable
14. Cache service and keyword info in service and keyword structures and use associative arrays to return info (unfortunate that can't have a scalar and array variable of the same name)
15. Use variable traces so can do KTL read or write simply by reading or writing the corresponding Tcl variable
16. Provide text window which can be used to enter Tcl commands and provide command recall in that (e.g. `tkCon`)
17. Catch signals (tclX has facilities for this)