1. Introduction

This document describes the tcl command and command types which are used to communicate with EPICS database servers. The application libraries upon which et is built include tcl, tk, tcl-dp, and blt. The reader of this document is assumed to be familiar with tcl/tk.

2. Running et

Start et by typing:

```bash
et_wish
```

In the extensions source directory of et (i.e. extensions/src/tcl_et) is a subdirectory examples containing examples using et. These examples together with the following descriptions of the pv command types should be used when learning to use et.

3. Command Syntax

The command syntax consists of three or more words, with the first three words having the same meaning in each command. The words after the first three words are dependent on the type of command issued. A command example follows:

```bash
pv link (ai1 ai2 ai3) (IOC:ai1 IOC:ai2 IOC:ai3)
```

- First word: the tcl pv command
4. pv Command Types

- Second word: the command type i.e. `link`. Command types which communicate with an EPICS database can be either synchronous (blocking) or asynchronous. A 'w' suffix (i.e. `linkw`) makes the command type blocking. The final word in blocking command types i.e. `linkw`, `getw`, and `putw` is a timeout value. The timeout value is optional and defaults to 10.0 sec.

- Third word: the name or list of names of tcl variable(s) i.e. `ai1` or `{ai1 ai2 ai3}`

- Other words: depend on the type i.e. for type `link` the fourth word consists of a name or list of names of EPICS database process variables i.e. `{BEAM:turnOn KLYmodAnode}`

Examples of all supported types:

```tcl
pv link {c0 ai1 ai2 ai3 ai4} (T:calc T:ai1 T:ai2 T:ai3 T:ai4)
pv get c0
pv getw ai1 11.1
pv put ai1
pv info {c0 ai1 ai2 ai3 ai4} (state status severity time units)
pv stat ai1
pv mon {c0 ai3 ai4} {puts "I can execute a script for you"}
```

<<<<start multi-element (vector) record oriented types>>>>>>>

```tcl
pv link (wf wfl) (vong:xy566WaveformCh0 vong:xy566WaveformCh1)
pv vdef (wf wfl) (0 256 5)
pv vset wf
pv vdis wf $graph line1
```

### 4. pv Command Types

**Syntax:**

```
pv link(w) tclVars iocVars <optional timeout>
```

**Description:** Establish a link at run-time between tcl variables and process variables in existing IOC databases. The tcl variables are automatically created during the operation. For string and enum process variables the tcl type created will be a string.* For all other process variable types the tcl variable type created is a double. A tcl variable may be subsequently linked with a different process variable (i.e. re-linked). When re-linked the old link is completely removed.

**Return:**

- 0 successful
- 1 unsuccessful and error is described in tcl variable “errorCode”

**Comments:** The user must always check the state of the link after `link` and `linkw` type commands, using either the 'stat' or 'info' command types. The state must be 'OK' (see `stat` and `info` type commands) before further operations on the tcl or process variable(s) are made. When the `link(w)` is first issued 'state' is set to 'IO'. If a link to a database process variable is successful the state changes to 'OK'. If the link was unsuccessful the state will be something other than 'IO' or 'OK', i.e. NC (Never Connected), LC (Lost Connection), RD (No Read Access), WR (No Write Access). For blocking links, i.e. `linkw`, the state can be checked immediately after the `linkw` command is issued. For non-blocking links, i.e. `link`,

* For ENUM database process variables which DO NOT have strings defined, use of numbers is supported.
4. pv Command Types

the state may still be in the 'IO' state after the link command is issued and subsequently
(asynchronously) change to another state. For the non-blocking command it is up to the user
code to wait until a 'state' transition or timeout has occurred. I recommend using blocking
links because of their coding simplicity.

Examples of correct forms of link(w):

```tcl
pv link ail IOC:beamVoltage
pv linkw (ail aol bil) (IOC:beamCurrent IOC:outputCurrent IOC:beamStatus)
set varList(aill aol bil)

set iocList (IOC:beamCurrent IOC:outputCurrent IOC:beamStatus)
pv link $varList $iocList
pv linkw $varList $iocList
if {[pv linkw ail IOC:beamVoltage] == 1) puts stdout $errorCode
```

get

Syntax:

```tcl
pv get(w) tclvars <optional timeout>
```

Description: Update the linked tcl variables to the current values stored in the IOC
databases.

Return:

- 0 successful
- 1 unsuccessful and error is described in tcl variable “errorCode”

Comments: The user should always check state of the link after get and getw type
commands using either the 'stat' or 'info' command types. The state must be 'OK'.
Additionally the severity of the process variable should be checked to, at least, make sure that
it is not INVALID. I recommend using the 'stat' command type for checking.

Examples of correct forms of get(w):

```tcl
set varList (ail aol bil)
set iocList (IOC:beamCurrent IOC:outputCurrent IOC:beamStatus)
pv link $varList $iocList

pv get $varList
pv getw $varList
pv get (ail aol)
pv getw (aol bil)
pv get bil
pv getw bil
```

put

Syntax:

```tcl
pv put(w,q) tclvars <optional timeout>
```

Description: Update the current process variable values stored in the IOC databases to the
linked tcl variable values. The put and putw types require notification from the IOC that
record processing has completed for all the linked variables in the command. The putw blocks
until all notifications have been received. The putq returns to the application as soon as the
proper requests have been forwarded to the IOC. I recommend using the putq type unless
there is a compelling reason for wanting to wait for record processing associated with a
process variable to complete.
4. pv Command Types

Return:

- 0 successful
- 1 unsuccessful and error is identified in tcl variable "errorCode"

Comments: User should always check state of the link after put, putw, and putq type commands. The state must be 'OK'. I recommend using the 'stat' command.

Examples of correct forms of put(w,q):

```tcl
set varList {ail aol bil}
set iocList {IOC:beamCurrent IOC:outputCurrent IOC:beamStatus}
pv link $varList $iocList

pv put aol
pv putw aol
pv putq aol
pv put {ail aol bil}
pv putw {ail aol bil}
pv putq {ail aol bil}
```

mon, cmon

Syntax:

```tcl
pv mon tclVars <optional script>
pv cmon tclVars
```

Description: Establish/Clear an EPICS database monitor for the linked tcl variable(s). The linked tcl variable is automatically updated whenever a monitored event is received. An optional tcl script, when defined in the command, will be executed whenever a monitored event is received from an IOC. Using the optional script, the user can implement an completely event driven application.

Return:

- 0 successful
- 1 unsuccessful and error is identified in tcl variable "errorCode"

Examples of correct forms of mon and cmon:

```tcl
set varList {ail aol bil}
set iocList {IOC:beamCurrent IOC:outputCurrent IOC:beamStatus}
pv link $varList $iocList
pv link {cO ail ai2 ai3 ai4} {T:calc T:ail T:ai2 T:ai3 T:ai4}

pv mon $varList
pv mon $varList {puts "Print this every event"}
pv mon $varList tclProcedureName
pv mon ail
pv mon ail {puts "Print this every event"}
pv mon ail tclProcedureName
pv mon {ail bil}
pv mon {ail bil} {puts "Print this every event"}
pv mon {ail bil} tclProcedureName
pv cmon $varList
pv cmon ail
pv cmon {ail bil}
```

info

Syntax:
**pv info tclVars requestedInfo**

**Description:** Obtain information in the form of a list of lists about linked tcl variables stored in the interface. Any or all of the following information may be requested in any order. Information is available for:

- **state:** of the communication link*
- **severity:** as defined by EPICS
- **status:** as defined by EPICS
- **time:** from IOC process variable shown as 10/03/94 14:04:36.791566783
- **units:** as defined in EPICS record
- **name:** of linked IOC process variable
- **choices:** for enum types
- **hpr:** as defined by EPICS
- **lpr:** as defined by EPICS
- **hihi:** as defined by EPICS
- **hi:** as defined by EPICS
- **lolo:** as defined by EPICS
- **lo:** as defined by EPICS
- **precision:** as defined by EPICS
- **ioc:** what IOC the process variable came from
- **access:** read & write privileges
- **size:** number of elements

**Return:**

- 0 successful
- 1 unsuccessful and error is identified in tcl variable “errorCode”

**Examples of correct forms of info:**

```
set varList (ail aol bil)
set iocList (IOC:beamCurrent IOC:outputCurrent IOC:beamStatus)
pv link $varList $iocList
pv link (c0 ail a12 a13 a14) {T:calc T:ail T:a12 T:a13 T:a4}
pv getw $varList

pv info ail state
pv info ail [state ioc name units severity status]
pv info ail (units state name ioc)
pv info $varList (state severity status time)
pv info $varList severity
pv info (ail bil) {name choices hopr lopr precision units lo lolo ioc access}
```

---

**stat**

**Syntax:**

```
pv stat tclVar associative_array_name
```

**Description:** Obtain state, status, severity, and time information about a single process variable as elements (state, status, severity and time) of an tcl associative array named in command. The array is created if it doesn’t exist.

* ‘state’ is a two-character string, whose possible values are:
  - NC->NeverConnected, LC->LostConnection, RD->NoReadAccess,
  - WR->NoWriteAccess, IO->I/OInProgress, OK->OK

---

**EPICS Release:** R3.12  **ET:** EPICS TCL/Tk Interface  **5**
4. pv Command Types

Return:

- 0 successful
- 1 unsuccessful and error is identified in tcl variable "errorCode"

Comments: In tcl scripts 'stat' provides a more convenient command type than 'info'.

Examples of correct forms of stat:

```tcl
set varList (ail aol bill
set locList (IOC:beamCurrent IOC:outputCurrent IOC:beamStatus)
pv link $varList $locList
pv link (c0 ail ai2 ai3 ai4) (T:calc T:ail T:ai2 T:ai3 T:ai4)
pv getw $varList

pv stat ail arrayName1
pv stat bi2 arrayName2
pv stat ai1 arrayName3

set arrayName1 (state)
set arrayName2 (severity)
set arrayName3 (status)
```

vdef, vset, vdis

Syntax:

```tcl
pv vdef tclVars (base size precision)
pv vset tclVars (list of values to be written to tclVars)
pv vdis tclVar graph graph_element
```

Description: All three of these command types are used to deal with multi-element and waveform records. For multi-element and single element records the link, get, put, and mon command types operate the same and transfer all the process variable elements into an internal et interface buffer. However, for multi-element process variables the linked tcl variable only reflects the value of the first element. The user reads/writes/displays multi-element data stored in the interface buffer using a "view" mechanism. A view is a defined subset of a multi-element record which will be subsequently used for accessing data from the record.

- vdef defines a view which specifies the base, size and number of significant digits. vdef can be defined for each linked tcl variable.
- vset is used either to return to tcl a list consisting of the values in the view with the number of significant digits defined by vset or to write into the interface the values defined by the final word (normally a list of values) to vset command type. The base and maximum number of values to be written are determined by the vset command type.
- vdis is used with the blt graph widget to display the process variable view as an "blt graph element" on a "blt graph".

Return:

- 0 successful
- 1 unsuccessful and error is identified in tcl variable "errorCode"

Examples of correct forms of vdef, vset, and vdis:

```tcl
set graph .g
blt-graph $graph -width 500
$graph element create line1 -symbol circle -bg red
```

*When a list of values to be written is not included as the final word of the vset command type, vset returns a list of values (defined by vdef).*
5. Usage

There are typically only three steps needed to communicate with an EPICS database:

- Establish a link between a tcl variable and a process variable (sometimes called channel) and check that the link has been established ok. This operation uses a link type and an info or stat type.
- Either update the tcl variable to reflect the current value of the process variable via a get type command or update the process variable to reflect the value of the tcl variable via a put type command and check that the update has completed ok. This operation uses a get or put type and a stat type.
- In the case of the get type operation, insure that the data received from the process variable is valid (i.e. not INVALID). This operation uses a stat/info type.

The user must keep in mind that command types link, get, and put involve transferring data over a network which means that the user must check that the network operation has successfully completed. The return result (either 0 or 1) of these command types together with the stat/info command type are used to insure that the operation was successful. In addition, when the command type is a get the user must insure that the data is valid (the network operation might have succeeded but the GPIB instrument timed out during a GPIB read). The stat/info command type is also used to check the validity of data.

The user must keep in mind that both the process variable in an IOC and the tcl linked variable can change at any time and are only synchronized only during a get or put type command.

Use of lists is encouraged in link, get, put and mon types to improve the efficiency of the network communications.

Use of mon types is encouraged for process variables which change infrequently (at least less frequently than referenced in script) and for applications which are designed to be event driven.

When checking the state, status, severity, and time in scripts, it is usually easier to use the stat command type. Info works well for interactive use.

The put and putw command types require notification from the IOC that the record has processed before completion. It is much better to rely on readback process variables to ensure that a put operation has successfully completed. With a readback available use of putq is preferred. The putq does not require IOC notification before completing.
When using asynchronous forms of `link`, `get` and `put`, the application must give the X-event loop time to process before each check on completion. The `tcl/tk` event loop processes whenever user code is not running, or whenever certain `tcl/tk` command are issued as described in the `tcl/tk` documentation. Otherwise, the application code will end up in an endless loop testing for something (state, status, severity, time) that never gets updated.

In the script associated with monitors, **THE SYNCHRONOUS COMMAND TYPES `LINKW`, `PUTW` and `GETW` SHOULD NOT BE USED** (this is a network software limitation related to code reentrancy).

---

**DISCLAIMER**

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.