

## Re: A subst-antial question

---

*Source:* <http://coding.derkeiler.com/Archive/Tcl/comp.lang.tcl/2006-04/msg01017.html>

---

- *From:* Neil Madden <nem@xxxxxxxxxxxxxx>
  - *Date:* Sun, 23 Apr 2006 16:05:19 GMT
- 

Busirane wrote:

Don Porter wrote:

```
% ::comm::comm send 51395 [list array set foo $pairs]
```

Let me restate the question. Suppose I have a script:

```
set script {  
array set foo $pairs  
set bar $somethingElse  
}
```

This would typically have been passed into a procedure instead of set literally, where "pairs" and "somethingElse" would be defined in the caller's context. I want to be able to run the same script either locally or remotely. No preprocessing is necessary to run it locally. I know I have to substitute variables locally before I can set them remotely, and a simple [subst] won't do it. I don't think [list]-ifying it in the caller's context, as you suggest, will work either, because it'd lose the whitespace command delimiters. I could process each line, substituting each word on the line and preserving it as a single word, then reassembling the substituted script, but is that the only way?

I may have oversimplified things in my original posting by using a single-line script... sorry!

There are a number of ways you could approach this. As you say, [subst] doesn't work as it expands variables into multiple words. If you do use subst, you'll also probably want the -nocommands switch, and possibly -nobackslashes. You could also try using [string map] to manually substitute each variable with a [list]-quoted value, e.g., something like:

```
% string map [list \ $foo [list $foo]] {  
stuff with $foo in it
```

## Re: A subst-antial question

```
}
```

stuff with {this is foo} in it

But that won't handle backslash escapes.

An alternative, in Tcl 8.5 (or 8.4 with the dict package), would be to not perform the substitutions locally, but instead to snapshot the environment of the script as a dict, and then send that along with the code (as a kind of closure) to be evaluated remotely using [dict with]:

```
proc capturelevel {{level 1}} {
  incr level
  set env [dict create]
  foreach name [uplevel $level { info locals }] {
    upvar $level $name var
    catch { dict set env $name $var }
  }
  return $env
}
```

```
proc sendscript {id script} {
  set env [capturelevel 1]
  comm send $id [list with $env $script]
}
```

Where, on the other end you have a proc:

```
proc with {__env__ __script__} {
  dict with __env__ $__script__
}
```

To use it, you'd do something like:

```
proc doStuff {} {
  set foo ...
  set bar ...
  sendscript 51395 {
    do stuff with $foo and $bar
  }
}
```

This has the advantage that [dict with] takes care of making sure that everything is substituted correctly. Of course, it does mean that you end up shipping the environment across the network, which could be expensive, depending on the size of the environment. If you need more than just locals, then you can adjust [capture] appropriately. You'll also need to add more code if you want to also handle arrays properly.

See <http://wiki.tcl.tk/15778> for some more background on this.

— Neil

.