Creating Custom Stars Derived from TclScript

```
defstar {
    name {MyFancyWidgetStar}
    derivedFrom {TclScript}
```

Add your own parameters:

```
    state {
        name{howManyWidgets}
        type{int}
        default{10}
    }
```

Hide the tcl_file parameter:

```
    setup {
        tcl_file = "~me/my_directory/myfile.tcl";
        tcl_file.clearAttributes(A_SETTABLE);
    }
```

Your parameters are accessible in your Tcl script:

```
    set n [set ${starID}(howManyWidgets)]
```

Another Example

```
# This procedure is invoked every time the star fires,
# thus providing a synchronized run
proc goTcl_$starID {starID} {

    # Suppose we know that the star has two inputs.
    # We can read them and add them together as follows.
    set inputVals [grabInputs_$starID]

    # Split the input list into individual elements
    set xin [lindex $inputVals 0]
    set yin [lindex $inputVals 1]

    # Add the numbers and send to the output
    setOutputs_$starID [expr $xin+$yin]
}
```

Note that Tcl is not a good way to do arithmetic, in general. Too slow. Also, the above star should iterate over however many inputs are actually connected.

Parameters

A global array with the name given by the value of the starID is created to store the following information:

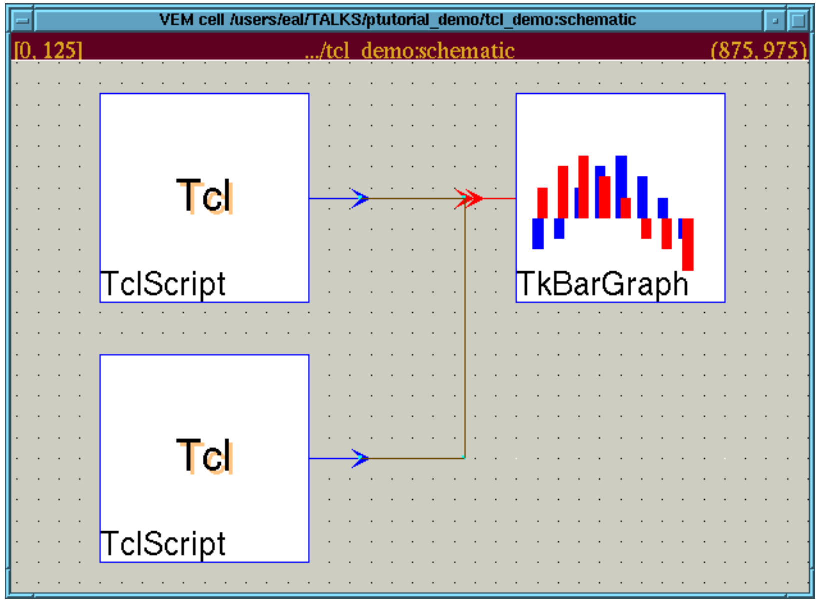
- **[set ${starID}(numInputs)]: The number of inputs actually connected to the star.**
- **[set ${starID}(numOutputs)]: The number of outputs actually connected to the star.**
- **[set ${starID}(tcl_file)]: The name of the Tcl file used by the star.**

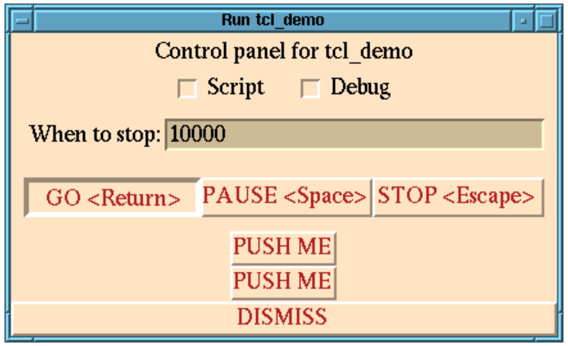
In addition, a set of procedures are provided for convenience and for uniformity of appearance:

- **ptkExpanEnvVar**
- **ptkImportantMessage**
- **ptkMakeButton**
- **ptkMakeEntry**
- **ptkMakeMeter**
- **ptkMakeScale**

Allowing Repeated Runs

```
set s $ptkControlPanel.middle.button_$starID

# Add the buttons only if they don't already exist
if {! [winfo exists $s]} {
    pack [button $s -text "PUSH ME"]
    bind $s <ButtonPress-1> "setOutputs_$starID 1.0"
    bind $s <ButtonRelease-1> "setOutputs_$starID 0.0"
}
unset s
```

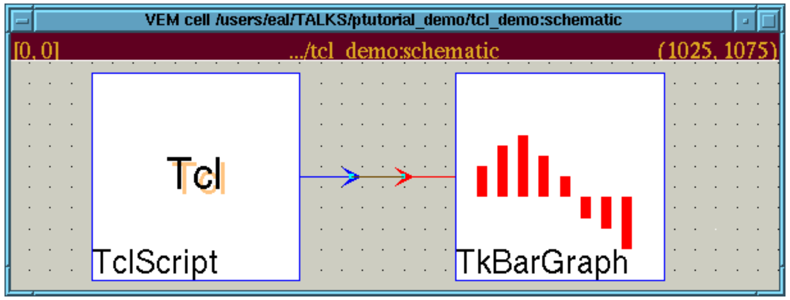


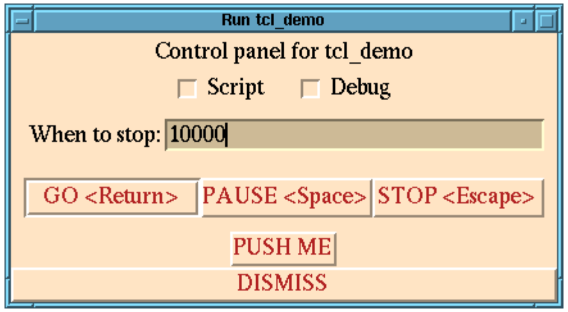


- buttons are only created if they don't already exist.

Putting the Button in the Control Panel

```
# Use a variable to store the long window name
set s $ptkControlPanel.middle.button_$starID

# Put a button in the window
pack [button $s -text "PUSH ME"]

# Bind an action to pushing the button
bind $s <ButtonPress-1> "setOutputs_$starID 1.0"
bind $s <ButtonRelease-1> "setOutputs_$starID 0.0"
```





Notes:

- no more name conflict (try using more than one instance).

An Awkward way to be Quasi-Object-Oriented

Global variables that are set before your script is sourced:

- **starID** (a unique identifier for your star)
- **ptkControlPanel** (the name of the control panel window)

Star procedures you can call from your script

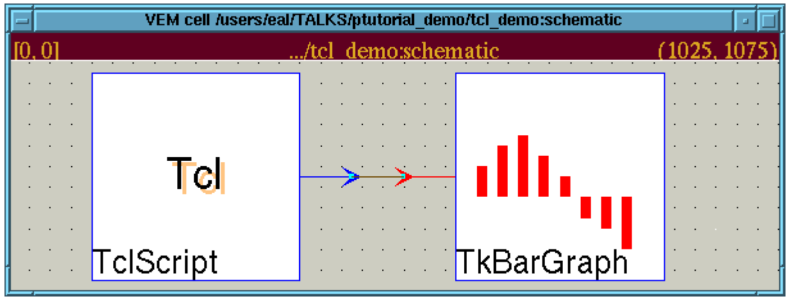
- **setOutputs_$starID**
- **grabInputs_$starID**

Procedures you can define in your script

- **goTcl_$starID**
- **wrapupTcl_$starID**
- **destructorTcl_$starID**

Example

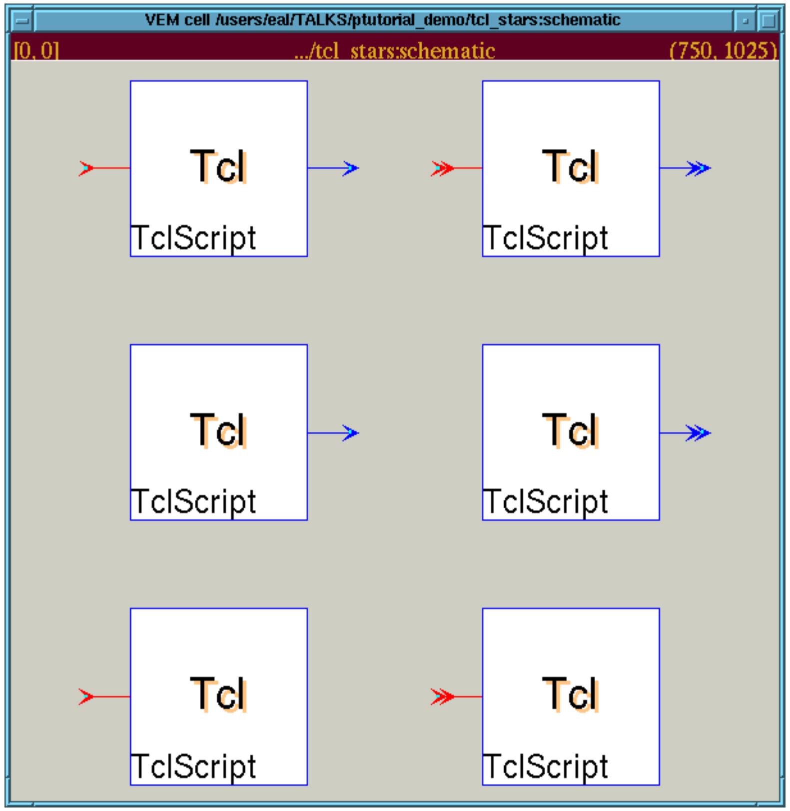
```
# Create an empty top-level window
toplevel .top

# Put a button in the window
pack [button .top.b -text "PUSH ME"]

# Bind an action to pushing the button
bind .top.b <ButtonPress-1> "setOutputs_$starID 1.0"
bind .top.b <ButtonRelease-1> "setOutputs_$starID 0.0"
```



Notes:

- you cannot use more than one instance of this script (name conflict).
- this script fails to remove the window it creates.
- the system runs free, unsynchronized to the button.

TclScript stars



The single parameter is the name of a Tcl file that is sourced at setup time. The behavior of the star is determined by this file.

Extending the GUI

Definitions:

- **Tcl is an interpreted “tool command language” designed by John Ousterhout at Berkeley.**
- **Tk is an associated X window toolkit.**

Essential reference:

J. Ousterhout, *Tcl and the Tk Toolkit*, Addison Wesley, Reading, Mass., 1994.

Tcl and Tk scripts may be invoked by stars and targets.