Using Tcl for Embedded Applications

Introducing Tcl as:

Language for Embedded Applications ...

... to which a GUI may be added with Tk

Dipl.-Ing. Martin Weitzel
Technische Beratung für EDV
http://tbfe.de

FPGA-Kongress 2016 Sprachen / L
2016-07-12 (16:30 ... 18:0

Agenda

1. What is Tcl?
2. How may Embedded Application use Tcl?
3. Tcl/Tk-Based Graphic User Interfaces
4. Code-Walking an Example
5. Questions and Discussion

You are welcome to interrupt the speaker with questions* and – es
during the code-walk –– propose to try small changes.

*: Your questions will be answered during the presentation to the best of the speaker's abilitie
in private communication after the presentation.

What is Tcl?

Tcl is short for

- Tool
- Command
- Language

It is the scripting language for a number of Design Tools ...

... but Tcl may well be used "Stand-Alone" too!

What does this mean?

Requirements

As an interpreted language Tcl needs ...

- (... guess what ...)
- **an Interpreter**

> The Tcl Interpreter is available as executable program for many f
> of Unix – of course including Linux and Mac OS X – and Micros
> Windows.

Obtaining Tcl

Moreover: Tcl is free software ...

- ... so you may use it without paying license fees.

And: The Tcl source code is available ...

- ... just in case you might need an interpreter for an "exotic pla

One of the major advantages of free software is that you do not d
on "strategic business decisions"* of an external company with re
to its products, i.e. whether such are

- considered to be part of their core competence and
  will therefore be supported for a long time, or
- determined to be obsolete and discontinued on short notice.

See also: https://www.tcl.tk/software/tcltk/

*: Which you might also spell as "desire and mood" or simply: whim.

And There are a Number of IDEs Too ...

If you want to do "*Serious Application Development*"* there are a n
of IDEs available too:

From the Community:

- ...
- ...
- ... (see: http://wiki.tcl.tk/998)
- ...
- ...

For Eclipse:

- http://www.eclipse.org/

For Microsoft Windows:

- http://www.activestate

---

*: Small rant caused by recent discussions about the "necessity" of IDEs for serious software d
Not denying IDEs may be a big boon to get an overall understanding of a complex piec
you have not written from scratch, and
granting the fact there are of course scenarios where the support of an IDE multiplies pro
If using vs. not using an IDE is taken as making the difference between "serious software d
vs. "just some toying around", then for the most part of his professional career
(a) the author of this never did any serious software development as ...
(b) **nobody else did throughout the 70s, 80s and part of the 90s**, when
(c) an editor plus the command line was all you had, and so it probably is
(d) a sheer wonder that all those major software systems developed during these ancient t
all! (Maybe just out of pure luck?)
But: "*Your mileage may vary*" as was the catchphrase in the good ol' days of the User
everybody who feels the need to shoot sparrows with canon balls is free to do so ... if this
the touch of "seriousness" to his or her day-to-day job.

Application Development Language Features

- Flow control to express the program logic
- Support for basic data structures to help organize related data
- Subroutines and maybe *Objects and Classes*[*] to provide struct
- Mechanisms to – build and use – libraries to get a good portion
  - Modularity and
  - Reusability

> Tcl has all of this ... and good set of add-ons and extensions t

Because:
Modularity and Library Support are worth so much the more if ther
add-ons available providing exactly – or at least "more or less" – th
features helpful for or required by the application under developm

---

[*]: With respect to OO-style programming Tcl offered some classical "add ons", like Incremer
XOTcl, or Snit.

Tcl for Which Kind Applications?

Tcl may be used (in embedded applications) for:

- High-Level Control Logic
- Prototyping*
- Testing

> More will be shown walking through excerpts from real Tcl co

*: There has been at least one case int the professional carer of the author when a piece o originally planned as prototype and "proof of concept" was sufficient as the final solution.

Tcl/Tk-Based Graphic User Interfa

Tcl's extension Tk is

- is **not** the latest-greatest super nifty and cool cutting-edge achievement in "advanced GUI design" ...
- ... but **it is**:
  - Mature and Stable
  - relatively light-weight
  - well sufficient for simple cases
- furthermore it
  - has a relatively low learning curve, but also
  - lends itself well to the experienced users*

More will be shown walking through excerpts from real Tcl co

*: Well, maybe with the addition: ... the experienced user *who is willing to learn* and – to so adapt to the design of Tk. If you take away some developer's beloved GUI library and try him or her to switch to Tk, you **may** find yourself running into a major barrier rejection and

Code-Walking Some Examples

Some of the examples also demonstrate interfacing Tcl with C/C++

Knowing this option exists may help to calm down worries Tcl mig

- … turn out **not** to be sufficient to fulfill (timing) requirements
- … leading to a complete restart, with …
- … throwing all code developed so far into the trash bin.

A Tiny Demo Application in Tcl

The demo application shown here will simulate a thermostate:

- A heater is switched on when the temperature drops below a t
  and
- it is switch off when the temperatur rises above another tresh

As the first step a prototype is developed in "pure Tcl".

Then the "thermostate logic" is implemented in C and tested using
TDD style as well as a Tcl-based GUI again.

The Tcl Prototype (1)

First there are two global variables defined, which – in "the real wo
will probably be somehow connected to a sensors and an actor:

```
set sensed_temperature 21.0
set system_heater 0
```

Two more globals represent the tresholds of the thermostate switc

```
set low_trigger 19.0
set high_trigger 21.0
```

The Tcl Prototype (2)

The thermostate's "switching logic" is in this subroutine:

```
proc switch_heater {temp} {
        global low_trigger
        global high_trigger
        global system_heater
        if {$temp <= $low_trigger} {
                set system_heater 1
        }
        if {$temp >= $high_trigger} {
                set system_heater 0
        }
}
```

In a real application this subroutine were called repeatedly or at le
every change of the sensed temperature changes.

The Tcl Prototype (3)

For the purpose of testing a Tk-based GUI is created, using a "slide
(`scale`) to simulated sensed temperature values:

```
proc create_gui {} {

        scale .system_temperatur\
                -orient horizontal\
                -from -20.0 -to 55.0 -resolution 0.5\
                -variable sensed_temperature\
                -command temperature_changed

        label .heater_state

        pack .system_temperatur\
                -side bottom\
                -fill x

        pack .heater_state\
                -side top\
                -expand 1\
                -fill both
}
```

The Tcl Prototype (4)

The (yet) missing link to the termostate logic is this – a (Tcl) sub-ro
called whenever the slider is moved:

```
proc temperature_changed {args} {
        global sensed_temperature
        switch_heater $sensed_temperature
        show_heater_state
}
```

As a reaction the thermostate logic would change the value of the
variable `system_heater`.

The Tcl Prototype (5)

Last and finally the value assigned to `system_heater` will be visuali
way ...

```
proc show_heater_state {} {
        global system_heater
        if {$system_heater} {
                .heater_state configure\
                        -text "ON"\
                        -background red
        } else {
                .heater_state configure\
                        -text "OFF"\
                        -background blue
        }
}
```

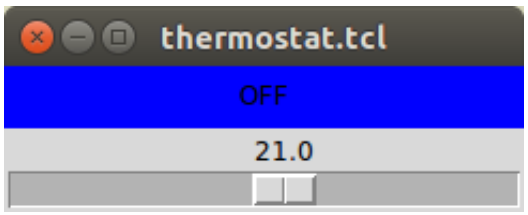
The Tcl Prototype (6)

... leaving this to start-up everything:

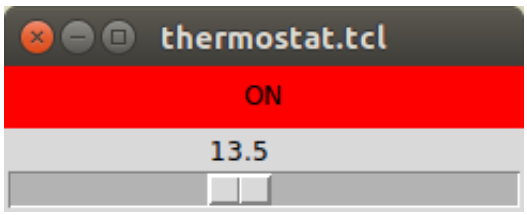
```
set sensed_temperature 21.0;     # assume this at the start
create_gui;                      # (as the name says) and
temperature_changed;             # show consistent state
```

Putting everything together, this is how the Tk-GUI finally looks:*

Heater switched off:



Heater switched on:



*: For the whole code as a single file see: Examples/Thermostate/Tcl_Prototype/thermostate.tc

A solution in Tcl plus C/C++

In this case the "core logic" switching the heater on and off is implemented in thermostate.c*

```
const float low_trigger = 19.0;
const float high_trigger = 25.0;
int system_heater = 0;

void switch_heater(float temp) {
        if (temp <= low_trigger) {
                system_heater = 1;
        }
        if (temp >= high_trigger) {
                system_heater = 0;
        }
}
```

With the (obvious) public interface in thermostate.h:

```
extern int system_heater;
extern void switch_heater(float temp);
```

---

*: Assume it does something more complex with a critical timing beyond the limits of Tcl.

A solution in Tcl plus C/C++ (2)

It may also have automated tests as this one in thermostate-test.c

```
#include <assert.h>
#include <stdio.h>

#include "thermostate.h"

int main() {
        system_heater = 0;

        switch_heater(20.0f); assert(system_heater == 0);
        switch_heater(19.5f); assert(system_heater == 0);
        switch_heater(19.0f); assert(system_heater == 1);
        switch_heater(17.0f); assert(system_heater == 1);
        switch_heater(20.0f); assert(system_heater == 1);
        switch_heater(24.9f); assert(system_heater == 1);
        switch_heater(25.0f); assert(system_heater == 0);
        switch_heater(30.0f); assert(system_heater == 0);
        switch_heater(3.0f);  assert(system_heater == 1);

        (void) printf("** ALL TESTS PASSED\n");
        return 0;
}
```


A solution in Tcl plus C/C++ (2)

Integrating the C/C++ code with Tcl is easy and and systematic wi
tool SWIG.

At first the interface needs to be described as in thermostate.i:*

```
%module thermostate
%{
#include "thermostate.h"
%}

int system_heater;
void switch_heater (float temp);
```

*: The similarity to a C/C++ header file is obvious and the original intent was that SWIG m
interface description from a C/C++ header file like the one already shown. But this would
SWIG is capable of parsing all complexities of the C++ syntax including advance
Furthermore, as frequently not all of what is contained in a header file constitutes the inter
code written in Tcl, some limited amount of code duplication apears to be acceptable.

A solution in Tcl plus C/C++ (3)

For the interface first a C-wrapper and finally a [Shared Object (.s
is created.

These are the relevant sections of the Makefile:

```
…
thermostate.so:\
        thermostate.o\
        thermostate_wrap.o
        gcc -shared thermostate.o thermostate_wrap.o -o$@

thermostate_wrap.o:\
        thermostate_wrap.c
        gcc -c -fPIC -I/usr/include/tcl8.6 thermostate_wrap.c

thermostate_wrap.c:\
        thermostate.i\
        thermostate.h
        swig -tcl thermostate.i
…
```

[Shared Object (.so-file)]:

A solution in Tcl plus C/C++ (4)

The remaining code in Tcl/Tk mainly needs to create the user inter
exactly as it was done in the prototype. Beyond that the actual so
provided in thermostate.tcl shows how an additional interface trac
any modification to the global `system_heater` variable can be estab
with Tcl's feature to [`trace`] access to variables:*

```
…
trace add variable system_heater {write} showvar

proc showvar args {
        global system_heater
        puts $system_heater
}
…
```

*: Extended systematic instrumentation of (Tcl-) code to support trace-logs is another topic
networking features might come in handy: instead of creating console output with `puts`
information might be written to a Tcp/Ip socket automatically connecting to a server t
output and also to filter it based on various criteria.

And More – If Time Allows

So you name it ...

... otherwise:

Thanks for Listening!

