Interfacing Tcl with the World (When Scripting is not Enough)

Dipl.-Ing. Martin Weitzel
Technische Beratung für EDV
64380 Roßdorf, Germany
www.tbfe.de

Prelude

- It all began in the late 90s ...
  - ...when I decided to declare my home an “MSfZ” (Microsoft free Zone)
  - There has been too much of frustration with Windows 3.x while Linux was on the verge to become a replacement
  - But often this excluded me from utility or convenience software supplied with electronic equipment
  - One such example is a multimeter I bought around the year 2000

continued ...

A True Story

...continued

- So what were my options?
  - Not using the nifty RS232 interface?
  - But this was part of the reason to buy that particular MM in the first place!
  - Retracting from my brave proclamation by using the MS-Windows software supplied with the MM?
  - Or try a Linux software for the MM myself?
- Luckily it turned out some brave soul already had tackled the hard part, i.e. analysed the wire protocol and published it on the internet

The wire protocol is both, “stupid simple” but rational to keep that little device “as dumb as possible”:

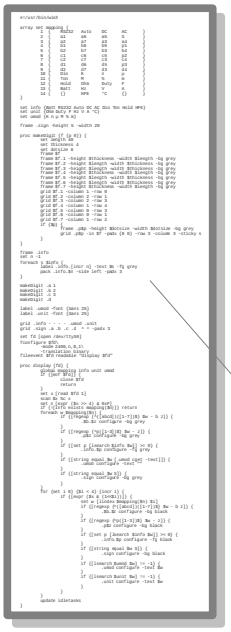
- Repeatedly transmitted are 14 bytes,
- each of which carries a sequence counter cycling from 1 through 14 in its high nibble …
- … while its low nibble codes the display state (i.e. whether some LCD digit segment or unit indicator is visible or not).

To be honest: to find out the latter might have taken me long, since I “naturally” expected a measuring value to be transmitted in ASCII (digits), or maybe BCD, or whatever ... but I'd never expected coded LCD digit segments!

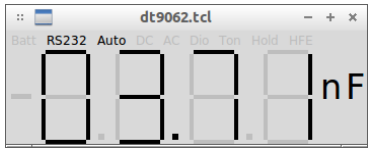
continued ...

A Multimeter Remote Display

...continued



- And here's the result of
  - a morning of Tcl programming fun and ...
  - ... an afternoon of cleaning up the result
  - (with no time left to add comments ☺)



Since you are not expected to read a 2.8 pt size font (but of course may give it a try ☺), here's the word-count output:

```
$ wc <dt9062.tcl
 126   561 4273
```

(i.e. one page of DIN A4, printing on front-side and back)

Typical Hardware Interfaces

- Prevalent interfaces are
  - Ethernet (most often used for classic TCP/IP, with maybe IPv6 on the rise, but actually open to many protocols)
  - USB (typically for a a serial data stream or presenting itself as a file storage volume)
  - RS232 (yes, it seems “the condemned live longer”)
- Still in use sometimes
  - SPP (Standard Parallel Port, now standardises by IEEE 1284, formerly “Centronics Interface”)

Hardware Interfaces of Embedded Devices

- Generally “I/O-Ports”
  - Used as “single bits” or in bit-groups of any size
  - Sometimes dedicated, sometimes combined and programmable
- Different electrical characteristics
  - Switching to supply voltage or ground ...
  - … maybe with a pull-up or pull-down resistor and sometimes even more special protection circuitry (e.g. for de-bouncing)
- A/D converters to take sensor measurements
- D/A converters for controlling various kinds of actors

“Software to Hardware” Interfaces

- Its good practice today to shield most idiosyncrasies of peripheral devices at the driver level
- Therefore at the application level there are much fewer abstractions to deal with
  - TCP/IP is most commonly used via the socket abstraction
    - In analogy to a plug and a (wall-mount) socket a connection may be available or in use
    - In the second case it presents itself as serial data stream
  - RS232 and SPP are typically “just serial data” streams too
  - USB may be a serial stream too or present a file system

“Software to Software” Interfaces

- Complex software systems are often structured into components that need to communicate with each other
- There are various ways to handle this
  - **Classic IPC:** Usually limited to a single node ans OS dependant in its details
  - **TCP/IP:** Local use is not uncommon as scaling is easy then by accumulating or distributing components over nodes
  - **Pipelines:** originally a prominent Unix feature – of course readily assumed by Linux – and of tremendous utility

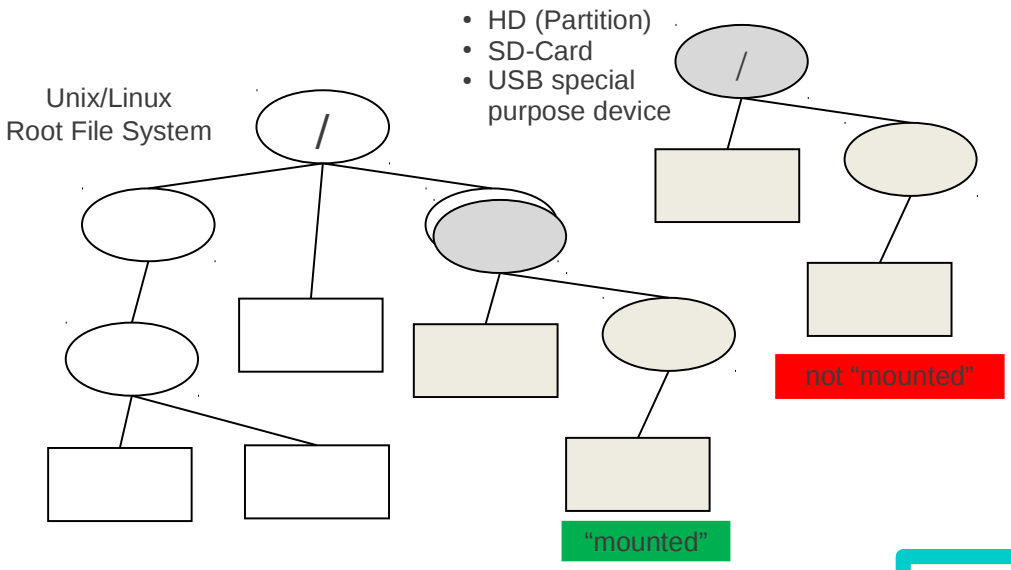
Communication from Tcl's Point of View

- Given proper driver support, “wiring” a Tcl application with divergent hardware components requires just to handle
  - Files and/or
  - Sockets
- In a design with separate software components, the handling child processes from Tcl may be an issue too
- Finally, by adding C/C++ modules to Tcl – a “dead easy job” when SWIG is used for the glue code – any requirement some whimsical piece of hard-/software might impose is satisfiable
- **If it can be done in C/C++, it can be done from Tcl too!**

Accessing File Systems

- A modern and very successful approach is to decorate a piece of hardware as if it were a part of the file system
  - The classic Unix *device file system* might have started it ...
  - … but surely *Plan 9 from Bell Labs* (a Unix successor) brought it to a first blow ...
  - … and so today it's an essential part of Linux too
- For a Tcl application the following commands are relevant:
  - `file` with many sub-commands for queries and operations on files systems and chosen files in entirety
  - `glob` to get an – optionally filtered – list or directory entries

Mounting File Systems



Streamed Data

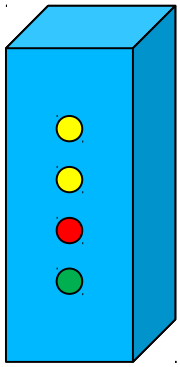
- Data streams in Tcl follow the classic file abstraction:
  - First use the Tcl command open to get an (opaque) handle
  - Then use this handle to
    - read data with
      - `gets` (full lines) or
      - `read` (portions of any chosen size)
    - write data with
      - `puts` (typically but not necessarily full lines)
    - and finally release used resources with `close`

Stored Data

- In addition to streamed data there is the option
  - (via the file handle)
    - to query the current position with the command `tell`
    - continue at a chosen position with the command `seek`
- Obviously this is close to the C/Posix model of file access
- If positions are determined by calculations, some care must be taken if translations are in effect
  - e.g. CR-NL → NL (or vice versa)
  - Opening “seekable” files in binary mode is to recommend

Fabricated Device Driver Example

- Controlling status LEDs via text output
  - Assignment: `A..D` from top down
  - Codes: `0` = off, `1` = on (steady), `2` = flash



Shell
```
echo –n A0B1C2D1 >/dev/keleds
```

Tcl
```
set fd [open /dev/keleds w]
puts $fd -nonewline A0B1C2D1
close $fd
```

C++
```
std::ofstream("/dev/keleds").write("A0B1C2D1", 8);
```

C/C++
```
int fd= open("/dev/keleds", O_WRONLY);
write(fd, "A0B1C2D1", 8);
close(fd);
```

Data Transmission vs. Sender/Receiver Synchronisation

- The key insight here is:
  - Transmitting any number of data bytes between a source and a sink often is the easy part ...
  - … with the more difficult challenge is to enable the sending and receiving end to tell or find out each others readiness
- Following the Unix model
  - Read and write operations are by default synchronous
  - I.e. the sender/receiver may be automatically suspended – without consuming CPU cycles – and resumed
  - With event driven designs as asynchronous counterpart

The Pipeline Abstraction

- The Pipeline abstraction – a prominent feature of early Unix – provides an elegant and efficient way to
  - Combine data transmission
  - with sender/receiver synchronisation
- A pipeline (aka FIFO)
  - Is associated with buffer for a certain amount of data
  - Suspends the receiver until data becomes available
  - Suspends the sender if the buffer space is filled

Classic Pipelines

- To set-up a classic pipeline on Unix (Linux) there must be
  - either a parent-child or child-parent relationship between sender and receiver
  - or both must be siblings, i.e. descendants from a common ancestor that prepared the connection
- Therefore the typical use of pipelines in Tcl applications is to
  - write data to a receiving process as its *standard input*
  - read data from a sending process as its *standard output*

Named Pipes

- To overcome the common ancestor limitation Unix System V added Named Pipes
  - Such have an entry in the file system
  - When opening a named pipe the rendezvous principle is applied, i.e. the process “arriving” first is resumed
    - A reader that comes first has to wait for a writer
    - A writer that comes first has to wait fro a reader
  - As soon as reader and writer are present, data exchange happens equivalent to a classic pipeline
  - **It is even possible to remove the file system entry then**

(Unix/Linux) Device Files

- Serial data streams sent or received through hardware interfaces are not different from any other streamed data
  - On Unix/Linux there is an entry in the `/dev`-directory
  - It might be named `/dev/ttyS0`, `/dev/ttyS1`, … (or `com`*X* on Windows) … but also completely different – RTFM!
- Same for USB interfaces giving access to serial data – including but not limited to USB-RS232 converters – except ...
  - … the device name may not be present as directory entry until the USB hardware is connected
  - … some more device specific set-up might be necessary, e.g. creating a “hot plug script” could become necessary

Configuring RS232 (Commonly Supported Options)

- Typically required transmission parameters can be set
  - when opening the device file: `open … -mode` *`spec`*
  - any time later: `fconfigure … -mode` *`spec`*
    - where *`spec`* is *`baud,parity,databits,stopbits`*
- More options may depend on hardware and/or driver, e.g.
  - Hardware flow control: `-handshake` *`type`*
    - where *`type`* is `none`, `rtscts`, `dtrdsr`, or `xonxoff`
  - Software flow control (*XON* / *XOFF*): `-xchar` *`xnxf`*
    - where *`xnxf`* is a list of the two characters sent for *XON* (enable sending) and *XOFF* (stop sending)

RS232 Advanced Usage

- Again **depending on** appropriate **driver / hardware support** ...
- **... output control signals**
  - *RTS*, *DTR* (hardware lines) and
  - *BREAK* (logical zero on data line for 250..500 msec)
  - may be generated (asserted) with
    - `fconfigure … -ttycontrol …`
- **... input control signals**
  - *CTS*, *DSR*, *DCD*, and *RI* (hardware lines)
  - may be queried with
    - `fconfigure … -ttystatus`

Using TCP/IP-Sockets in Tcl

- Tcl provides
  - Server sockets waiting in the “half-open” state …
    - … until a connection request comes in …
    - … triggering a previously registered handler …
    - … handing over a file handle which represents ...
    - … a bidirectional stream connection with the client
  - Client sockets to initiate a connection to a server ...
    - … returning of a file handle which represents …
    - … a bidirectional stream connection with the server

Providing and Using Web-Interfaces in Tcl

- Providing a Web-Interface to a Tcl application means:
  - Provide a server socket and then …
  - … “talk HTTP” over the bidirectional connection that is eventually created
- Using a Web-Interface in an Tcl application means:
  - Initiate a socket connection and then …
  - … “talk HTTP” over the bidirectional connection that is eventually returned
- **All in all: you have to know a bit of HTTP and little of Tcl!**

Database Access in Tcl

- TDBC is the generic interface and Tcl 8.6 is shipped with support for
  - MySQL
  - ODBC
  - PostgresSQL
  - SQLite
- You will also find specific Tcl extensions for *Oracle* (oratcl), *Informix* (isqltcl), *Adabas* (AdabasTcl)...

Spawning Child Processes

- The Tcl command `exec` spawns a child process (not necessarily implemented in Tcl), then by default
  - Arranges to catch the child process' output
  - Suspends the calling Tcl parent until the child terminates
  - Finally
    - delivers the child's standard output via its return value (which the caller may access by putting the exec command in square brackets)
    - or issues an error (which the caller may handle via a `catch` command, if the child ended abnormally)

Reading Standard Output from a Child Process

- The `open … r` command behaves special if the file name argument starts with a vertical bar (`|`):
  - The remainder of the file name argument then is considered to be an external command that will subsequently be started
  - The file handle returned is the read-end of a classic pipeline
  - The other end is connected to the child's standard output
- The Tcl application then runs concurrently with the child and may read the pipe
  - Asynchronously by registering a handler with `chan event`
  - Synchronously – by simply using `gets` or `read`

Writing to Standard Input of a Child Process

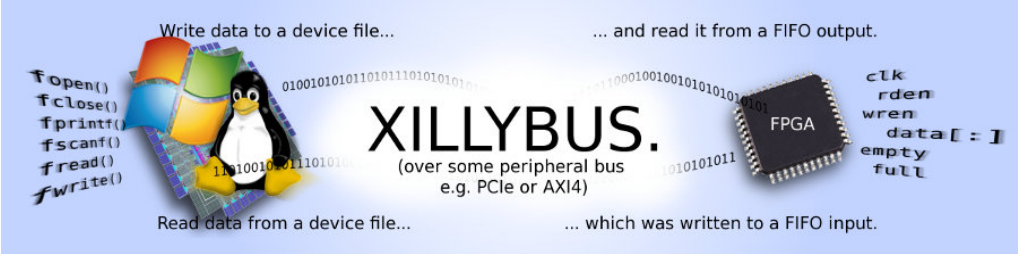
- The `open … w` command behaves special if the file name argument starts with a vertical bar (`|`):
  - The remainder of the file name argument then is considered to be an external command that will subsequently be started
  - The file handle returned is the write-end of a classic pipeline
  - The other end is connected to the child's standard input
- The Tcl application then runs concurrently with the child and
  - typically writes to its pipe end synchronously with `puts`, but
  - might be suspended until the child catches up reading

Adding C/C++ Modules Using SWIG

- Adding a new command implemented in C/C++ requires to:
  - Register the command (name and entry point) in Tcl's lookup-table
  - Provide some “clue code” to convert ...
    - … between what the Tcl provides or expects and ...
    - ... the command parameter types as defined in C/C++
- SWIG is a tool to create the registration and glue code
  - Details are based on an *interface description* …
  - … with a very familiar look to any C/C++ programmer
- **Once more: if you can do in in C/C++ you can do it in Tcl!**

Xillybus – No Plug!

- See http://xillybus.com/
  - **Looks actually very promising to a Unix veteran!**
  - But until today I have only flicked through the docs
  - **Currently I have not any practical Xillybus experience!**



Epiloque

- There once was a Tcl enthusiastic MSfZ proclaimer ...
  - … who had bought a multimeter with an RS232 interface ...
  - … wrote some “remote display software” for just for fun …
  - … and (mostly) forgot about it
- Ten years after … (no, not the rock giants are referred to here)
  - … that person tought a Tcl course to FPGA programmers …
  - … thought that Tcl software would make a nice example …
  - … had to spend nearly an hour to find the serial cable …
  - … and finally (ruffle – rataplan – drum roll) ...

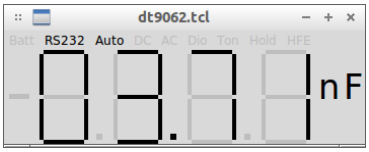
continued ...

The MM Remote Display Software Still Worked Flawlessly

- **Without changing a single line of code**

...continued

  - though written long ago
  - for a substantial earlier version of Tcl and Tk
  - back then on a Linux/Windows release as of a decade ago



... that trusty ol' Tcl Horse!

- (now: go and try this with some other old windows software ☺)

That‘s All

Any (more) Questions?

Thank You for Participating