

AndroWish – 963 days later

or

the Good,



the Bad,



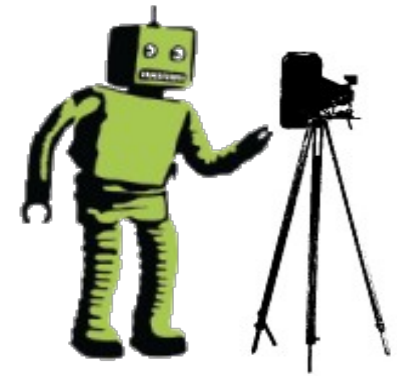
and

the Ugly.



borg camera . . .

- Tcl interface to deal with `android.hardware.camera`
- Capture images into Tk photo images since this is the most universal format
- Operations to start and stop continuous capture for live video display
- Operation to take a real photo with higher resolution in JPEG format
- Report availability of a new captured (live) image by virtual event `<<ImageCapture>>` to toplevel windows
- Report availability of a captured JPEG image by virtual event `<<PictureTaken>>`
- Facility to control properties/parameters of the camera(s)



borg camera commands (overview)

Minor command	Description
close	Close currently open camera
current	Return index of currently open camera (usually 0=rear, 1=front)
grayimage	Retrieve last captured image in gray scale
image	Retrieve last captured image in RGB
info	Retrieve current camera orientation relative to screen
jpeg	Retrieve last captured image as JPEG byte array
mirror	Flip captured image horizontally and/or vertically
numcameras	Return number of cameras
open	Open camera given index (usually 0=rear, 1=front)
orientation	Rotate captured image by 0, 90, 180, or 270 degrees
parameters	Get and/or set camera parameters (e.g. image size)
start	Start image capture
stop	Stop image capture
takejpeg	Initiate capture of a JPEG image (with higher resolution)

borg camera (a small Webcam #1)

```
proc init {} {
    image create photo img -width 640 -height 480
    borg camera open
    borg camera parameters preview-size 640x480 \
        picture-size 640x480 jpeg-quality 80
    bind . <<ImageCapture>> {
        borg camera image img
    }
    borg camera start
    pack [label .label -image img]
    socket -server request 8080
}
```

borg camera (a small Webcam #2)

```
proc request {sock args} {
  chan configure $sock -translation binary -blocking 0 \
    -buffering none
  after 100
  catch {chan read $sock 1000} err
  chan configure $sock -blocking 1
  if {![borg camera takejpeg]} {
    chan close $sock
    return
  }
  bind . <<PictureTaken>> [list send_jpeg $sock]
  chan puts -nonewline $sock "HTTP/1.0 200 OK\r\n"
  chan puts -nonewline $sock \
    "Connection: close\r\nContent-Type: image/jpeg\r\n\r\n"
}
```

borg camera (a small Webcam #3)

```
proc send_jpeg {sock} {  
    bind . <<PictureTaken>> {}  
    catch {  
        chan puts -nonewline $sock [borg camera jpeg]  
    }  
    catch {chan close $sock}  
    borg camera start  
}
```

```
init
```

zbar/dmtx commands

- Decode barcodes, QR codes etc. from photo images or byte arrays
- If the Tcl core supports threads, decode process can be carried out asynchronous, i.e. in the background using true concurrency



Examples:

```
zbar decode nameOfPhotoImage
```

returns a list of number of milliseconds for decoding, symbology literal, and result as byte array

```
zbar async_decode nameOfPhotoImage callback
```

carries out decoding asynchronously and invokes callback with arguments like in the synchronous example above

zbar/dmtx commands

Documentation on www.androwish.org:

[dmtx command](#)

[zbar command](#)



Examples:

[androwish:///assets/dmtx0.1/demos/android_demo%2Ehtml](#)

[androwish:///assets/zbar0.1/demos/android_demo%2Ehtml](#)

These links would work on an Android tablet, if this document were HTML and viewed in Firefox or Chrome

Emojis (Unicode 8.0)

```
package require tkpath

pack [::tkp::canvas .c -width 400 -height 400 -background white]

.c create ptext 200 200 -fontfamily Symbola -fontsize 70 \
  -fill gray70 -stroke black -strokewidth 1 -textanchor c \
  -text "\U1F601\U1F602\U1F603\n\U1F604\U1F605\U1F606\n\U1F607\U1F608\U1F609"
```



- Inspired by Jan Nijtman's Unicode 8.0 presentation at EuroTCL 2015
- Currently, two approaches are possible in Tcl core
- `TCL_UTF_MAX=4`: `Tcl_UniChar` is a 16 bit data type, codepoints beyond BMP are expressed as surrogate pairs
- `TCL_UTF_MAX=6`: `Tcl_UniChar` is a 32 bit data type, all codepoints correspond to a single `Tcl_UniChar`
- AndroWish chose `TCL_UTF_MAX=6` since this fits font rendering with AGG/freetype which take 32 bit codepoints

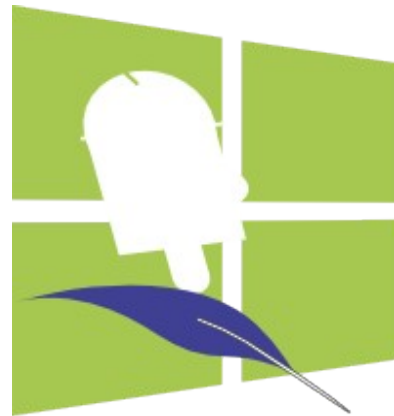
Emojis, things to consider

- AndroWish might work with `TCL_UTF_MAX=4`, but this is untested
- On Win32 many OS interfaces use the `WCHAR` data type (16 bit) thus an additional/another translation has to take place when dealing with the OS
- Text input on both Android (SDL2) and Win32 translate key events to 16 bit quantities with surrogate pairs
- Text input in X11 and MacOSX is still terra incognita and needs further investigation
- Font mapping in Tk/SDL, Tk/WIN32, and Tk/X11 (Xlib, not Xft) is memory hungry regarding subfonts
- Need for a comparison of run-time costs of `TCL_UTF_MAX=6` vs. `TCL_UTF_MAX=4`

tkpath and pdf4tcl

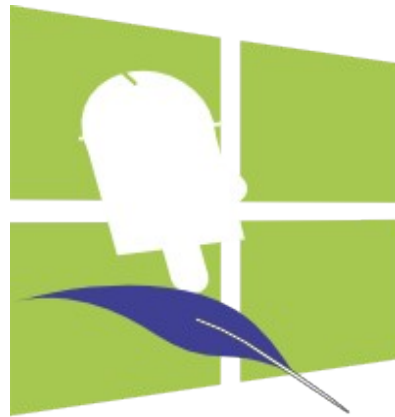
- With René Zaumseil's help: tkpath items output PDF primitives directly to pdf4tcl, conventional canvas items still use the pdf4tcl approach
- pdf4tcl now has additional interfaces for tkpath (for text output, image objects, extended graphics states)
- tkpath PDF generation calls into pdf4tcl
- Almost all tkpath item properties implemented, i.e. alpha transparency, gradient fills, images with tinting
- Unfinished: text/font w.r.t. encodings, rendering, and font substitution, but basic Latin 1 with standard fonts works
- Unfinished: repeating gradient fills

undroidwish



AndroWish sans the borg – a project just for pun

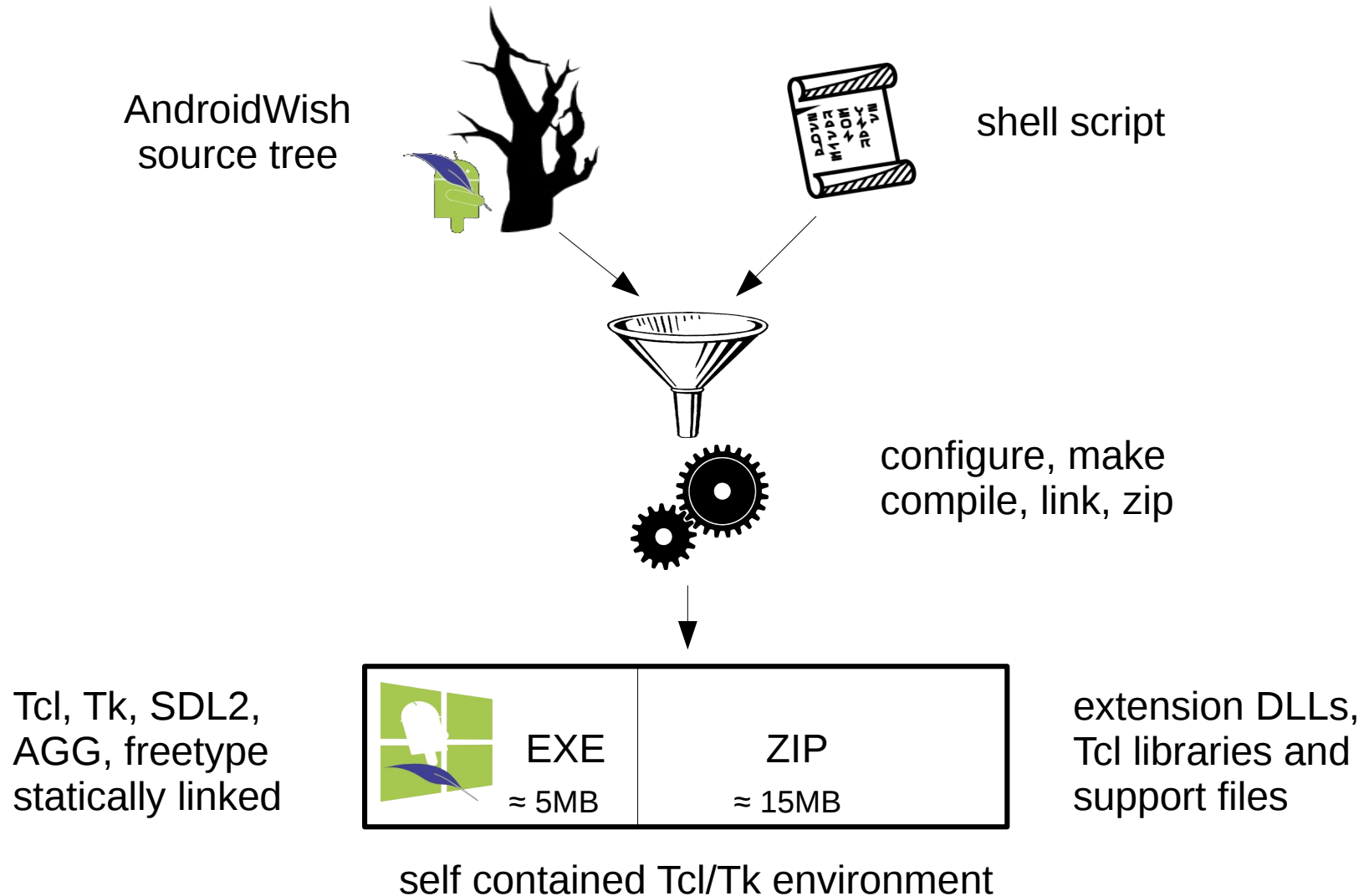
undroidwish



~~AndroWish sans the borg – a project just for pun~~

Wishful thinking for underdogs

undroidwish (making of)



undroidwish

- AndroWish source tree plus few extras
- Renders like AndroWish using SDL2/AGG/freetype
- Unicode 8.0 enabled (using `TCL_UTF_MAX=6` like AndroWish)
- Shell script to control `configure/make/install`
- For Linux target: standard compiler (`gcc`)
- For Win32/64 targets: MinGW64 cross compiler
- As in AndroWish many packages and extensions are built in
- Binary `undroidwish{,.exe}` is self contained
- No installation required, all packages and extensions in ZIPFS
- Alternative build using native windowing system (X11 or GDI) available (called `vanillawish{,.exe}`)

undroidwish, DLL hell avoidance etc.

- undroidwish (i.e. the Tcl/Tk executable part) are compiled and linked statically
- C++ code can introduce dependencies on libstdc++, fortunately, AGG can avoid this by using a custom memory allocator based on `Tcl_Alloc()` instead of `new`
- MinGW64 relies on MSVCRT only, thus Win32 undroidwish'es should work on Windows XP or newer
- For Linux, use a distro with older glibc for building, e.g. RHEL/CentOS 5, this allows to generate binaries which run on most Linuxen of the last 8 years
- Where more recent (read unsupported on older systems) DLLs are required, use runtime linking like `Tcl_LoadFile()`

undroidwish, continued

Documentation (rudimentary)

<http://www.androwish.org/index.html/wiki?name=undroidwish>

List of packaged extensions and libraries

<http://www.androwish.org/index.html/wiki?name=Batteries+Included>

Downloads of pre-built undroidwish'es for Windows and Linux

<http://www.androwish.org/download/index.html>

TWAPI in undroidwish

- Emmanuel Frécon: Wouldn't it be nice to have TWAPI in undroidwish?
- Tcl Windows API: a nontrivial Windows specific extension originally requiring MSVC tools for building
- Now available in Win32/64 undroidwish'es
- Built using MinGW64 cross compiler into a single DLL
- Ashok P. Nadkarni added support for Unicode 8.0 based on `TCL_UTF_MAX=6` in a few days (big thank you!)
- WITS (Windows Inspection Tool Set) included

v4l2 (and tcluvc)

- “Video for Linux 2” Tcl interface to video devices, e.g. webcams
- Tcl command very similar to “borg camera . . .”
- tcluvc similar to v4l2, interface to UVC type USB cameras (still early alpha state)
- v4l2 devices returns list of video devices from /dev directory or udev information
- v4l2 listen . . . establishes callback which is triggered by udev when USB cameras are plugged/unplugged
- v4l2 open . . . opens a video devices and establishes a callback to be invoked when an image was captured
- In the callback v4l2 image . . . transfers the captured image into a normal Tk photo image
- Images are captured and converted to RGB thanks to libv4l2 and its built in converters

tclwmf (WIP)

- Tcl interface to Windows Media Framework to use video capture devices, e.g. webcams
- Tcl command very similar to “v4l2”
- `wmf devices` returns list of video devices pair wise as symbolic link (WMF terminology) and friendly name
- `wmf open . . .` opens a video device given symbolic link name and establishes a callback to be invoked when an image was captured
- In the callback `wmf image . . .` transfers the captured image into a normal Tk photo image
- Images are captured in NV12 or YUY2 format and internally converted to RGB for `Tk_PhotoPutBlock()`

Questions?