AndroWish

- Native Tcl/Tk 8.6 port for Android ≥ 2.3.3 on ARM and x86 processors. Top goal: execute existing Tcl/Tk scripts on Android without modification.

- Uses Tim Baker's SDLTK project for graphics rendering, see http://members.shaw.ca/tnbaker/SDLTk
  - X11 emulation based on AGG (Anti-Grain-Geometry, http://www.antigrain.com) and SDL 2.0 (http://libsdl.org)
  - freetype font engine (http://www.freetype.org)

- Mounts its constituting APK (Android Package, i.e. the app) using a built-in ZIP file system based on mmap(2)

- “Batteries included” like TclKits (i.e. many ready-to-use Tcl extensions already bundled)

- Tcl commands available to use specific Android facilities
AndroWish, the big picture
AndroWish, limitations

- Due to Android process start up with respect to the window system `exec(n)` is limited to non-Tk processes, i.e. you can't `exec wish`.
- The bandwidth of device screen resolutions is broad (140 dpi ... 500 dpi) compared to usual desktop systems. But many elements of Tk widgets are pixel based.
• Partial Xlib replacement to implement rendering of standard Tk widgets using SDL and AGG and to support most features of extensions (treeview, tktable, TIX, BLT).

• Event handling by translating SDL events to X11 events plus additional virtual events
  - App life cycle (<<WillEnterBackground>>, <<WillEnterForeground>>, etc.)
  - Accelerometer is handled by SDL and mapped to an SDL joystick, and translated to Tk virtual event <<Accelerometer>>
  - Multi-touch events <<FingerUp>>, <<FingerDown>>, <<FingerMotion>>, <<PinchToZoom>>

• Other SDL goodies available through sdltk command

• Can be built standalone on other platforms (currently tested on Linux x86/Raspberry with X11 and DirectFB), see http://www.ch-werner.de/sdltk
- Obtain power management information (sdltk powerinfo)
- Control accelerometer (sdltk accelerometer)
- Control screensaver (sdltk screensaver)
- Show/hide virtual keyboard (sdltk textinput)
  - built into standard bindings of entry and text widgets
- Control emulation of middle and right mouse buttons for context menus and panning/scrolling (sdltk touchtranslate)
borg [ˈbɔrɡk]

- German imperative of “borgen” (to borrow)
- Inspired by Scripting Layer for Android (SL4A) and PhoneGap (now Apache Cordova)
- Junction to connect the (native) Tcl/Tk with the (Java) Android universe
- Use Android facilities from Tcl scripts
  - Information about device
  - Activities (Android apps)
  - Alarms, Notifications
  - Content (Android databases)
  - Location (GPS)
  - Text to speech and speech recognition
- Report device events using Tk virtual events (network status, location updates, sensor events)
Obtain information about the display

```
borg displaymetrics

density 1.33125 densitydpi 213 \ 
width 800 height 1216 \ 
xdpi 216.17021 ydpi 215.31126 \ 
scaleddensity 1.33125
```
Use Android's text-to-speech facility for speech output

```
borg speak "resistance is futile" \ en_US
```
Add an icon to Android's desktop, which launches a Tcl script contained in AndroWish's ZIP mounted /assets directory

```plaintext
borg shortcut add \nfile://assets/sdl2tk8.6/demos/widget
```
Use other Android activities (parts of other apps). Example: read a bar code using the http://code.google.com/p/zxing bar code scanner app.

```bash
proc barcode_read {code action uri type cat data} {
    array set result $data
    if {[info exists result(SCAN_RESULT)]} {
        # that is the barcode
        # result(SCAN_RESULT_FORMAT) is the barcode format
    }
}

borg activity com.google.zxing.client.android.SCAN \ 
    {} {} {} {} {} {} barcode_read
```
borg virtual events

- **<<LocationUpdate>>**: location information is available and can be read using
  
  ```
  borg location get
  ```

- **<<SensorUpdate>>**: a sensor can be read using
  
  ```
  borg sensor get ...
  ```

- **<<NetworkInfo>>**: network status was updated and can be read using
  
  ```
  borg networkinfo
  ```

- **<<Bluetooth>>**: Bluetooth status was updated and can be read using
  
  ```
  borg bluetooth state
  ```
Other AndroWish goodies

- `bluetooth` command allows to create Bluetooth client and server sockets providing the serial port profile (SPP). These sockets are normal Tcl channels.

- `usbserial` command allows to use certain USB to serial converters (FTDI, Prolific) similar to normal Tcl serial channels, i.e. `fconfigure` can be used to control the baud rate etc.
# Anatomy of AndroWish's APK

<table>
<thead>
<tr>
<th>Directory within APK</th>
<th>Description</th>
<th>Size uncompressed</th>
</tr>
</thead>
<tbody>
<tr>
<td>assets/</td>
<td>Application auxiliary files, Tcl libraries</td>
<td>38 MByte</td>
</tr>
<tr>
<td>tcl8.6/</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sdltk8.6/</td>
<td></td>
<td></td>
</tr>
<tr>
<td>fonts/</td>
<td>Default fonts, Deja Vu</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>“Batteries included” (SQLite, tcllib, tls, treectrl, ...)</td>
<td></td>
</tr>
<tr>
<td>app/</td>
<td></td>
<td></td>
</tr>
<tr>
<td>lib/</td>
<td>Native code, shared libraries</td>
<td></td>
</tr>
<tr>
<td>armeabi/</td>
<td>... for ARM processors</td>
<td>12 MByte</td>
</tr>
<tr>
<td>x86/</td>
<td>... for x86 processors</td>
<td>18 MByte</td>
</tr>
<tr>
<td>res/</td>
<td>Android Resources</td>
<td>few kByte</td>
</tr>
<tr>
<td>drawable/</td>
<td>... icons</td>
<td></td>
</tr>
<tr>
<td>layout/</td>
<td>... layout, styles</td>
<td></td>
</tr>
<tr>
<td>resources.asrc</td>
<td>Application resources</td>
<td>few kByte</td>
</tr>
<tr>
<td>AndroidManifest.xml</td>
<td>Application descriptor</td>
<td>few kByte</td>
</tr>
<tr>
<td>classes.dex</td>
<td>Compiled Java code for Android's Dalvik VM</td>
<td>&lt; 1 MByte</td>
</tr>
<tr>
<td>META-INF/</td>
<td>JAR Manifest, signatures of all files in APK</td>
<td>&lt; 1 MByte</td>
</tr>
</tbody>
</table>
Batteries included
How to roll your own app

● It is possible to re-use AndroWish's infrastructure (/assets directory with Tcl runtime, native shared libraries) from other apps.

● A slightly modified Java glue is required (about 300 kByte compressed Java classes, compared to ≥ 24 MByte of the complete AndroWish APK).

● Due to APK building (various Android tools and Java jarsigner needed) this must be done using Android's SDK and optionally Eclipse.

● A demo project using this approach is in the source tree of AndroWish (subdirectory hellotcltk)
Dive into the hive: Preparations

0. AndroWish's APK is installed and launched.

1. The static constructor of AndroWish's activity loads various shared libraries including `System.loadLibrary("main");`

2. The `JNI_OnLoad()` function of `libmain.so` remembers a pointer to the JVM among other initialization steps.

3. The constructor of AndroWish's activity calls a static native method in `libmain.so`, which keeps a Java object reference on the activity for later. The activity object is the main entry point to carry out Android-specific functions.
Dive into the hive: let the borg bark

1. Tcl evaluates borg beep.

2. The native code in libmain.so invokes a static method on the activity object obtained during startup:

```c
JNIEnv *env = GetJNIEnv();
jmethodID mid = (*env)->GetStaticMethodID(env, jactivity, "beep", "()V");
(*env)->CallStaticVoidMethod(env, jactivity, mid);
```

3. In the activity class the JVM executes this static method:

```java
public static void beep() {
    Runnable beeper = new Runnable() {
        public void run() {
            Uri ringuri = RingtoneManager.getDefaultUri(RingtoneManager.TYPE_NOTIFICATION);
            Ringtone tone = RingtoneManager.getRingtone(mSingleton.getBaseContext(), ringuri);
            if ((tone != null) && (!tone.isPlaying())) {
                tone.play();
            }
        }
    };
    mSingleton.runOnUiThread(beeper);
}
```

4. The notification sound is played back (more or less barking due to microscopic speakers).
1. Tcl evaluates the command `borg sensor enable 42` (that is our fictional sense of life sensor).

2. The native code in `libmain.so` invokes a static method, which switches that sensor on.

3. The AndroWish activity includes a supplemental class, which implements a `SensorEventListener` interface. When new sensor data is available, this method is invoked:

   ```java
   public void onSensorChanged(SensorEvent event) ...
   ```

   It stores sensor data in a synchronized field of the supplemental class. Finally this native method is called:

   ```java
   mAandroWish.nativeTriggerSensor(42);
   ```

4. This carries out the following code in `libmain.so`:

   ```c
   SDL_Event event;
   event.type = SDL_USEREVENT;
   event.user.code = 42;
   event.user.data1 = (char *) "SensorUpdate";
   if (SDL_PeepEvents(&event, 1, SDL_ADDEVENT, 0, 0) <= 0) return -1;
   return 1;
   ```

5. The SDL_USEREVENT is processed like other SDL events (touchscreen, mouse, keyboard) in the Xlib emulation layer and transformed into a virtual Tk event `<<SensorUpdate>>`, which is sent to all Tk toplevel windows.

6. Zero or more Tcl event handlers bound to `<<SensorUpdate>>` evaluate `borg sensor get 42` in order to read out the sensor data from the synchronized field in the supplemental class.
Activities come in two forms: one-shots without synchronization and a kind of remote procedure calls delivering results back to the caller. Execution is always asynchronous, i.e. the results are returned through callback functions. borg activity supports both modes.

1. borg activity ... -callback ... invokes native code in libmain.so, which translates the (many) parameters to their Java counterparts and invokes a static method in the AndroWish activity object which starts the activity.

2. This is remembered in a callback structure within libmain.so to match with the result later and to transport it back to Tcl.

3. The Android system carries out the activity and returns the result to the AndroWish activity object's onActivityResult() method.

4. In onActivityResult() the result is preformatted and presented to the native method nativeIntentCallback() of libmain.so.

5. In nativeIntentCallback() the matching callback structure is looked up, the preformatted results are translated to be appended to the -callback argument within the callback structure. That structure is finally queued to a thread-specific callback queue and a single byte is written into the write end of the pipe associated with the queue.

6. The file event handler on the read end of the pipe (running in the thread, which issued borg activity in step 1.) is invoked due to readability of the pipe. It empties the pipe and invokes all callbacks of the callback structures read out from the queue.

7. One-shot activities omit steps 2. to 6.

8. For speech recognition results the same mechanism (callback structure, queue, and pipe) is used.
X11 Emulation: The Puzzle
X11 Emulation: The Pieces

• SDL: device events, low-level timer, Android glue, frame buffer, low-level drawing surfaces, screen refresh functions.

• AGG: anti-aliased rendering (e.g. XDraw*() functions).

• freetype: low-level font support (loading TTF files, basic rendering).

• Event thread: transformation of SDL events to X events, screen updates from frame buffer (rate limited to 50 fps), event translation (pinch-to-zoom, middle/right mouse button emulation).

• Display: X11 display structure used to transfer X events to Tcl threads (one Tcl thread opens one Display like in Tk on X11).

• Big Xlib lock: Mutex plus condition variable to serialize X requests and to deal with server grabs like a real X server does.

• Pipes: OS level unidirectional communication channels to indicate queue not-empty state on display's X event queue similar to Tk on X11.
X11 Emulation: Claims & Reality

- If the behavior of the emulation is near real X11 not much additional code is needed in Tk.
- Porting existing Tk extensions becomes easy.
- Window background Pixmaps, clipping by Pixmaps, and rarely used GC functions are not implemented.
- Window manager functions are a bare minimum.
- Design inherently does not scale well on multiple cores.
20 minutes into the future

Tcl/Tk is quite easily portable and SDL already runs on many platforms. So let's think of ...

Windows \(\geq\) XP
Windows RT
Linux \(\geq\) 2.6 (X11 & DirectFB)
Mac OS X \(\geq\) 10.5
iOS \(\geq\) 5.1.1
Android \(\geq\) 2.3.3
(Free|Open|Net)BSD
Solaris
Haiku
PSP
Mir
Wayland

(list of more or less supported platforms/window systems on SDL 2.0.3)
Thank you.
Questions?