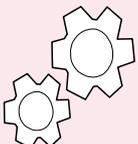


AndroWish  
meets

the  
ingeniously delightful  
Internet  of Things

# Bluetooth 4.0

- aka Bluetooth Smart, Bluetooth Low Energy, BLE
- supported on Android  $\geq 4.3$
- rapid build-up of simple links
- to communicate with sensors/actors
- designed to have very low power requirements
- builds on Generic Attribute Profile (GATT)

# Generic Attribute Profile (GATT)

- Service: collection of characteristics
- Characteristic: attribute containing a single logical value (e.g. temperature) described by zero or more descriptors
- Descriptor: attribute(s) describing a characteristic
- Discovery: facility to obtain a list of all services, characteristics, and descriptors of a device
- Notification: optional property of a characteristic to send unsolicited message on data change or periodically
- Scanning: detection of remote BLE devices with their friendly (human readable) name

# Generic Attribute Profile (GATT) (cont.)

- Objects (services, characteristics, descriptors) are identified by 128 bit UUIDs and carry certain meta data like read/write type, data type, permissions etc.
- Some descriptors are predefined, e.g.

00002902 - 0000 - 1000 - 8000 - 00805F9B34FB

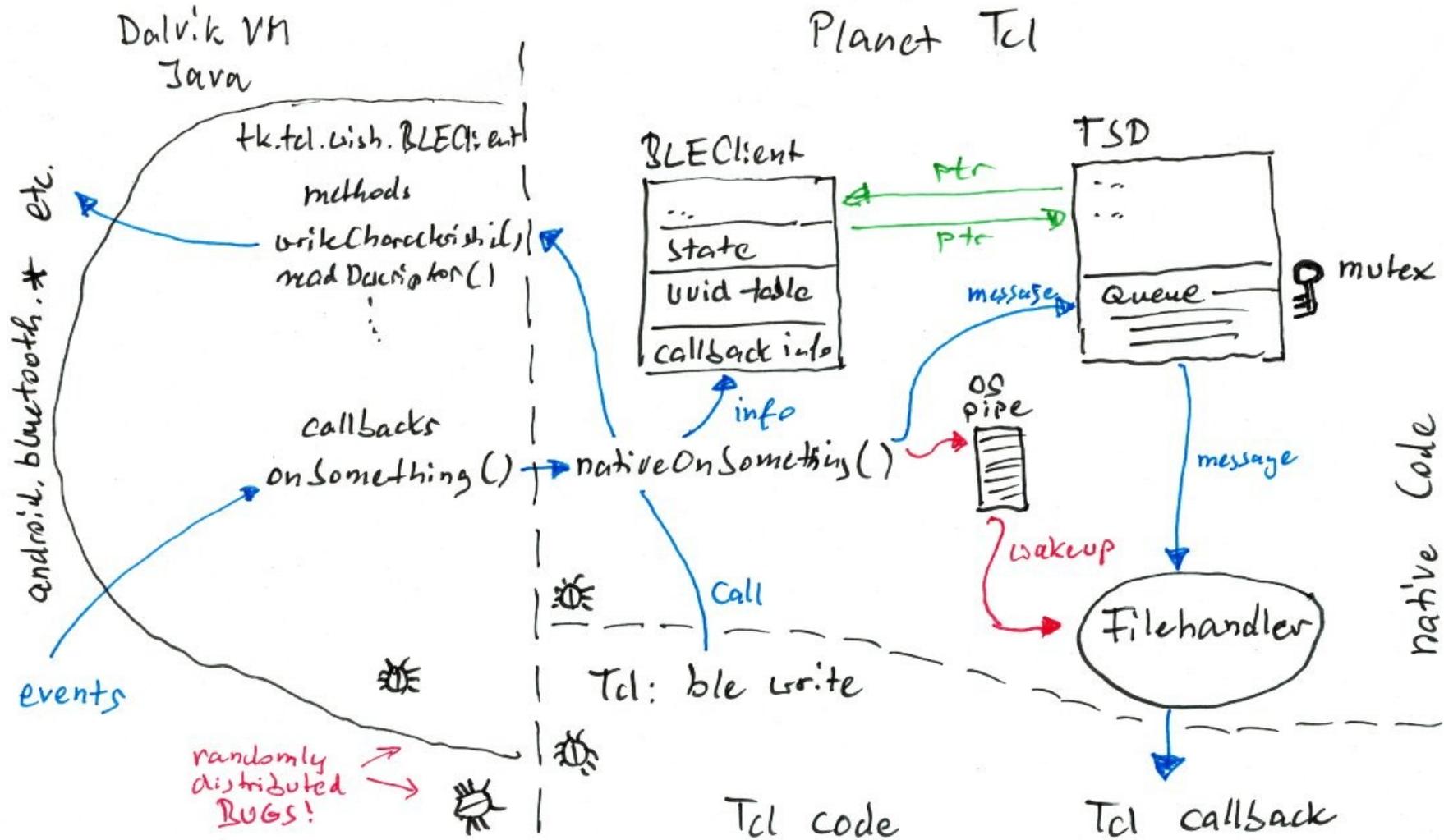
to enable or disable notifications by writing the 16 bit value 0x0001 or 0x0000 in little endian format, respectively.

- A rule to abbreviate UUIDs: write the first 32 bits or second 16 bits of the 128 bit UUID (00002902 or 2902 in the example above)

# Android BLE framework

- `android.bluetooth.BluetoothAdapter`: class to deal with the local Bluetooth interface, provides a callback for results of scanning for remote BLE devices
- `android.bluetooth.BluetoothDevice`: represents a remote Bluetooth device (which can be a BLE type device)
- `android.bluetooth.BluetoothGatt`: provides the facilities to connect to and to communicate with BLE devices
- `android.bluetooth.BluetoothGattService`: represents a GATT service
- `android.bluetooth.BluetoothGattCharacteristic`: represents a GATT characteristic
- `android.bluetooth.BluetoothGattDescriptor`: represents a GATT descriptor
- `android.bluetooth.BluetoothGattCallback`: an abstract class to report GATT events back to the application

# AndroWish's ble command



# AndroWish's ble command

- Connection setup and data exchange is event driven and asynchronous.
- Right after logical connection setup to a BLE device an automatic discovery is performed by the Java glue in order to learn the services, characteristics, and descriptors of the BLE device.
- In contrast to Android's `android.bluetooth.*` classes there's a single callback for all types of events which receives the event type as a single word and a dictionary with data depending on the type of the event, e.g.

```
proc callback {event data} { ... }
```

- A read operation is asynchronous, i.e. schedules the read. Actual data is reported in the callback.
- A write operation is asynchronous, too, i.e. the completion of the write is reported in the callback.

# ble minor commands (overview)

Minor command	Description
abort/begin/execute	Handling of write transactions
close	Close a BLE handle (for both, connection and scanner)
connect	Connect to a BLE device returning a connection handle
disable/enable	Enable and disable notifications of a characteristic
disconnect/reconnect	Disconnect and reconnect to BLE device
dread/dwrite	Read and write descriptors
read/write	Read and write characteristics
scanner	Obtain a BLE handle for remote device scanning
start/stop	Start and stop scanning for remote devices
info/callback	Obtain information on BLE handle(s)
userdata	Arbitrary user data associated with BLE handle
getrssi	Get remote signal strength indication of BLE device
services/characteristics/descriptors	Obtain information on device services, characteristics and descriptors
equal/expand	Operations on UUIDs

# b1e command (documentation)



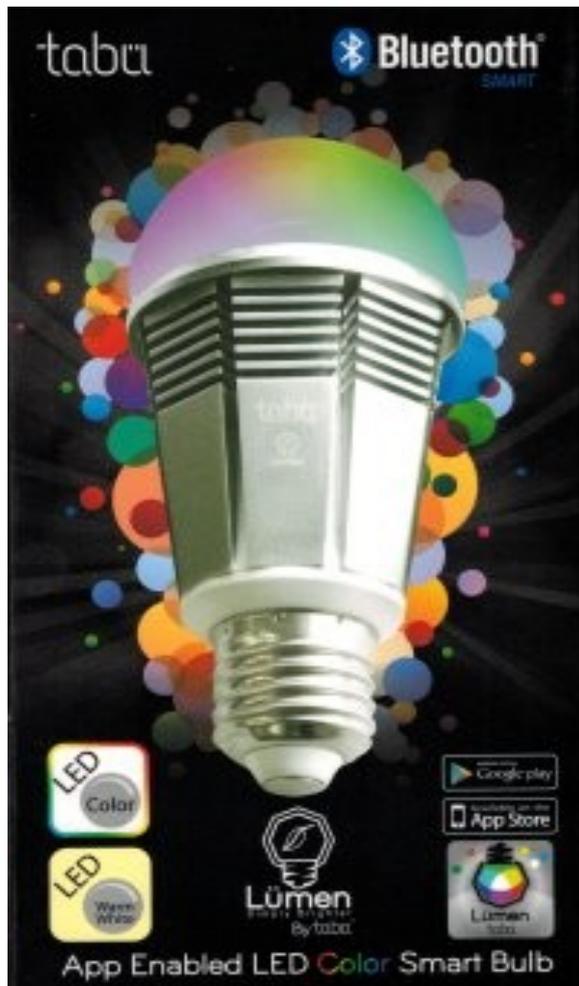
A man page for the b1e command in AndroWish can be found on

<http://www.androwish.org/index.html/wiki?name=b1e+command>

# ble command (costs)

- Java glue code (`tk.tcl.wish.BLEClient`) needs about 12 kByte Java byte code
- Native code (implementation of the `ble` command in C) needs about 12 kByte machine code (ARM) and 21 kByte machine code (x86)
- Total costs: about 45 kByte uncompressed

# Steampunk: the Smart Bulb



- LED color bulb controlled over Bluetooth Low Energy
- CMYK color model
- various built-in presets incl. “Disco” mode
- lamp is controlled by a single characteristic consisting of about 16 byte of data
- full demo available in AndroWish's source tree as

.../assets/ble1.0/demos/lumen.tcl

# Detect the bulb

```
proc scan {event data} {
  if {$event eq "scan"} {
    dict with data {
      if {[string match "iSmartLight*" $name]} {
        # found it, connect to it
        ble connect $address connect_step_1
        # close the scanner handle
        ble close $handle
      }
    }
  }
}

ble start [ble scanner scan]
```



# Connect the bulb (step 1)

```
proc connect_step_1 {event data} {
  if {$event eq "connection"} {
    dict with data {
      if {$state eq "connected"} {
        # connection setup magic in a write transaction
        ble begin $handle
        set magic1 [binary format H* \
          "08610766a7680f5a183e5e7a3e3cbeaa8a214b6b"]
        ble write $handle FFF0 0 FFF1 0 $magic1
        set magic2 [binary format H* \
          "07dfd99bfddd545a183e5e7a3e3cbeaa8a214b6b"]
        ble write $handle FFF0 0 FFF1 0 $magic2
        ble execute $handle
        ble callback $handle connect_step_2
      } elseif {$state ne "discovery"} {
        # fallback to scanning
        ble close $handle
        ble start [ble scanner scan]
      }
    }
  }
}
```

# Connect the bulb (step 2)

```
proc connect_step_2 {event data} {
  if {$event eq "transaction"} {
    dict with data {
      # trigger initial read of value
      ble read $handle FFF0 0 FFF1 0
      ble callback $handle connected
    }
  } elseif {$event eq "connection"} {
    dict with data {
      if {$state ne "connected"} {
        # fallback to scanning
        ble close $handle
        ble start [ble scanner scan]
      }
    }
  }
}
```

# Callback when connected

```
proc connected {event data} {
  if {$event eq "characteristic"} {
    dict with data {
      if {[string match "*FFF1-*" $cuuid]} {
        # store value in handle's userdata for later
        ble userdata $handle $value
      }
    }
  }
  } elseif {$event eq "connection"} {
    dict with data {
      if {$state ne "connected"} {
        # fallback to scanning
        ble close $handle
        ble start [ble scanner scan]
      }
    }
  }
}
}
```

# Turn the bulb on or off

```
proc bulb {on} {
  # we should have only one handle at any one time
  set data [ble info [ble info]]
  dict with data {
    if {$state eq "connected"} {
      set value {}
      binary scan [ble userdata $handle] H* value
      if {[string length $value] > 0} {
        if {$on} {
          set value [string replace $value 0 9 "01dfd99bb5"]
        } else {
          set value [string replace $value 0 1 "00"]
        }
      }
      set value [binary format H* $value]
      if {[ble write $handle FFF0 0 FFF1 0 $value]} {
        # trigger read back of value
        ble read $handle FFF0 0 FFF1 0
        # done, success
        return 1
      }
    }
  }
}
# not done
return 0
}
```



clock format [clock seconds] -format "%Q"

The mission: build a Tricorder



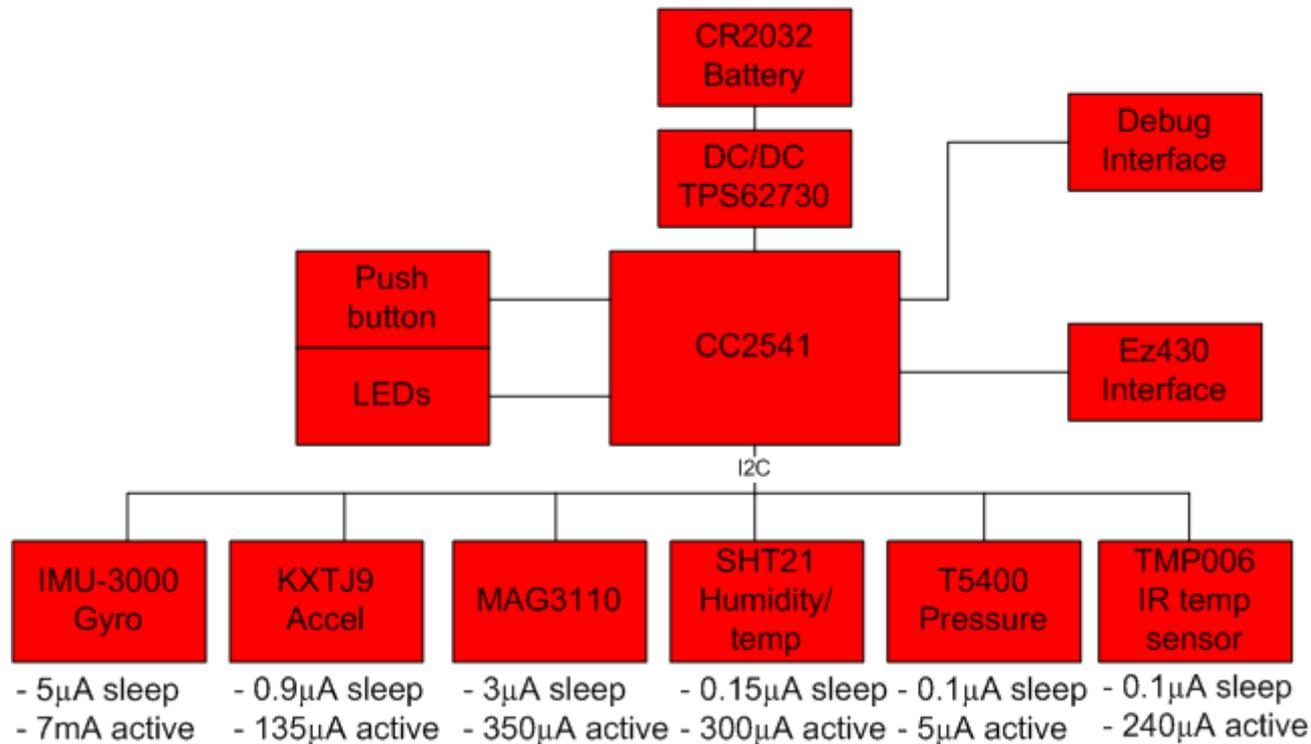
# Tricorder sensor component

## Texas Instruments CC2541 SensorTag Development Kit



- SoC based on 8051 MCU with integrated Bluetooth LE connectivity
- many sensors added on the PCB: IR temperature, humidity, pressure, accelerometer, gyroscope, magnetometer

# SensorTag block diagram



Source: [http://processors.wiki.ti.com/index.php/SensorTag\\_User\\_Guide](http://processors.wiki.ti.com/index.php/SensorTag_User_Guide)

# SensorTag UUIDs (excerpt)

Sensor	UUID	Format
IR Temperature	AA01 (value) AA02 (config)	2 * 16 bit little endian 1 * 8 bit
Accelerometer	AA11 (value) AA12 (config)	3 * 8 bit 1 * 8 bit
Humidity	AA21 (value) AA22 (config)	2 * 16 bit little endian 1 * 8 bit
Magnetometer	AA31 (value) AA32 (config)	3 * 16 bit little endian 1 * 8 bit
Barometric Pressure	AA41 (value) AA42 (config)	2 * 16 bit little endian 1 * 8 bit
Gyroscope	AA51 (value) AA52 (config)	3 * 16 bit little endian 1 * 8 bit
Buttons	FFE1 (value)	1 * 8 bit

# Enabling sensors and notifications

Snippet shows how ble enable commands for characteristics having notification property are accumulated during discovery.

```
...
characteristic {
  if {($state eq "discovery") && ($properties & 0x10)} {
    set cmds [ble userdata $handle]
    lappend cmds [list ble enable $handle $suuid $sinstance $cuuid $cinstance]
    ble userdata $handle $cmds
  }
  ...
}
```

Most sensors need to be enabled explicitly by writing sensor dependent commands in a configuration characteristic.

```
...
connection {
  if {$state eq "connected"} {
    set cmds [ble userdata $handle] ;# enable all notifications
    if {$cmds ne {}} {
      set cmd [lindex $cmds 0] ; set cmds [lrange $cmds 1 end] ; {*}$cmd
      # Add commands to turn various sensors on. Barometer needs two configurations
      # to load its calibration. Gyroscope has a bitmask for various axes.
      set on1 [binary format H* "01"]
      set on2 [binary format H* "02"]
      set on7 [binary format H* "07"]
      foreach {suuid cuuid on} { AA00 AA02 on1 AA10 AA12 on1 AA20 AA22 on1
        AA30 AA32 on1 AA40 AA42 on2 AA50 AA52 on7 AA40 AA42 on1 } {
        lappend cmds [list ble write $handle $suuid 0 $cuuid 0 [set $on]]
      }
      # Read barometer calibration.
      lappend cmds [list ble read $handle AA40 0 AA43 0]
      ble userdata $handle $cmds
    }
  }
  ...
}
```

# Process sensor value

Snippet shows how the magnetic field sensor value is converted.

```
...
switch -glob $cuuid {
  F000AA31-* {
    set x 0
    set y 0
    set z 0
    binary scan $value s1s1s1 x y z
    set ::sensortag(magnetic_x) \
      [format "%.5f" [expr {0-$x*2000.0/65536.0}]]
    set ::sensortag(magnetic_y) \
      [format "%.5f" [expr {0-$y*2000.0/65536.0}]]
    set ::sensortag(magnetic_z) \
      [format "%.5f" [expr {$z*2000.0/65536.0}]]
  }
  ...
}
```



Full demo available in AndroWish's source tree as

```
.../assets/ble1.0/demos/tricorder.tcl
```



Thank you.

Questions?