



# TuxDroid Java API - Documentation

Le 21 septembre 2007, à 13:6 par Ulhume...

## Object hierarchy

The all API is available through one simple object, `Tux`.

First you need to create and connect your Tux object

```
Tux tux = new Tux('localhost') // if your daemon is on an other machine,  
change localhost to its name  
tux.connect()
```

API allows you to control your TuxDroid from it's different body parts :

- `tux`. You can control speaking, turning left and right.
  - `tux.head()`. You can listen head button on this object.
    - `tux.head().eyes()`. This to control eyes opening/closing and IR emitting. You can also listen Eye movements and IR reception. No motor listening yet.
      - `tux.head().eyes().left()`. This to controls Blue LED. No LED listening yet.
      - `tux.head().eyes().right()`. This to controls Red LED. No LED listening yet.
    - `tux.head().mouth()`. You can control mouth movement here, and also listen to mouth events. No motor listening yet.
  - `tux.wings()`. You can control wings movement here. No motor listening yet implemented.
    - `tux.wings().left()`. Here to listen left wing switch.
    - `tux.wings().right()`. Here to listen right wing switch.

## Threading model

Every object can have commands and events. Commands allways return a Query object that you can use to block the call or not.

```
// Blocking call. Eyes will not be opened if wings are not  
Query query = tux.wings().open()  
query.waitForResponse()  
tux.eyes.open()  
  
// Non-Blocking call. Eyes and Wings will be open at (nearly) the same  
time.  
tux.wings.open()  
tux.eyes.open()
```

own thread. So if you push the head button and quickly after the right wing. Both call will occur at the same time, in tow separate threads.

## Movements

Basically, all parts that have motors, have methods `open`, `close`, `isOpen`, `toggle`. Parameters depends of hardware abilities. For example wings got speed and counter, eyes got counter but no speed.

```
// I gonna fly !!
tux.head().eyes().open()
tux.wings().toggle(3,Speed.veryQuick)
```



All movements are blocking calls for now. This because you can't use Eyes and Mouth at the same time. This is basically because they both use the same motor. Latter I'll allows wings And (Eyes or Mouth).



Be careful when you close Tux Eyes. As IR receptor is located here, you'll not be able to control Tux with the remote once closed. Or you'll have to be very close to go through.

When you want to move the all tux, you can use Tux object itself

```
// let's dance !
tux.turnLeft(3,Speed.slow)
tux.turnRight(2,Speed.quick)
```

## Lights

Lights are only available on Eye object. Methods are simply `lightOn` and `lightOff`.

```
// Lets be a cyclope
tux.eyes().left().lightOn()
tux.eyes().left().lightOff()
```



Calling those methods is non-blocking.

## Listening to events

Listening is basically the same as Swing model with some little calling differences. You always have :

- An XXXEventListener interface you can implements.
- An helper class BasicXXXEventListener that is an empty implementation of the previous interface for quick inheritance.
- A XXXEvent object given to every XXXEventListener methods.
- A XXXEventListeners object hosted by target body part object, generally called XXXListeners, to add/remove your listener.

For now there is 3 listeners : ButtonEventListeners that you can add to head, left wing et right wing. ShutterEventListener for eye and mouth. And RemoteControlEventListener for eyes.

```
// I'll dance if you tap on my head
tux.head().buttonListeners().add(new BasicButtonEventListener()
{
    public void buttonPressed(ButtonEvent event) throws TuxException
    {
        tux.wings().toggle(6, Speed.verySlow)
        tux.wings().toggle(6, Speed.veryQuick)
    }
})
```

Remote Control use the same concept :

```
// I control my Tux
tux.head().eyes().remoteControlListeners().add(new
BasicRemoteControlEventsListener()
{
    public void remoteButtonPressed(RemoteControlEvent event) throws
TuxException
    {
        switch (event.key())
        {
            case left
                tux.turnLeft(1, Speed.quick)
                break
            case right
                tux.turnRight(1, Speed.quick)
                break
        }
    }
})
```

## Voice

For now it's a basic but working voice implementation. There is no control on voice Yet but you can make Tux speak (with a tiny cute voice 😊 in a blocking or non-blocking way.

```
tux.head().mouth().open() // We don't want to wait here
tux.speak ("Hello world").WaitForResponse() // We wait tux finish
speaking
tux.head().mouth().close()
```

## That's all folks !

That's it for now, more will come soon 😊