

Pleo and the LifeOS




Introduction to Programming

Portland Area Robotics Society



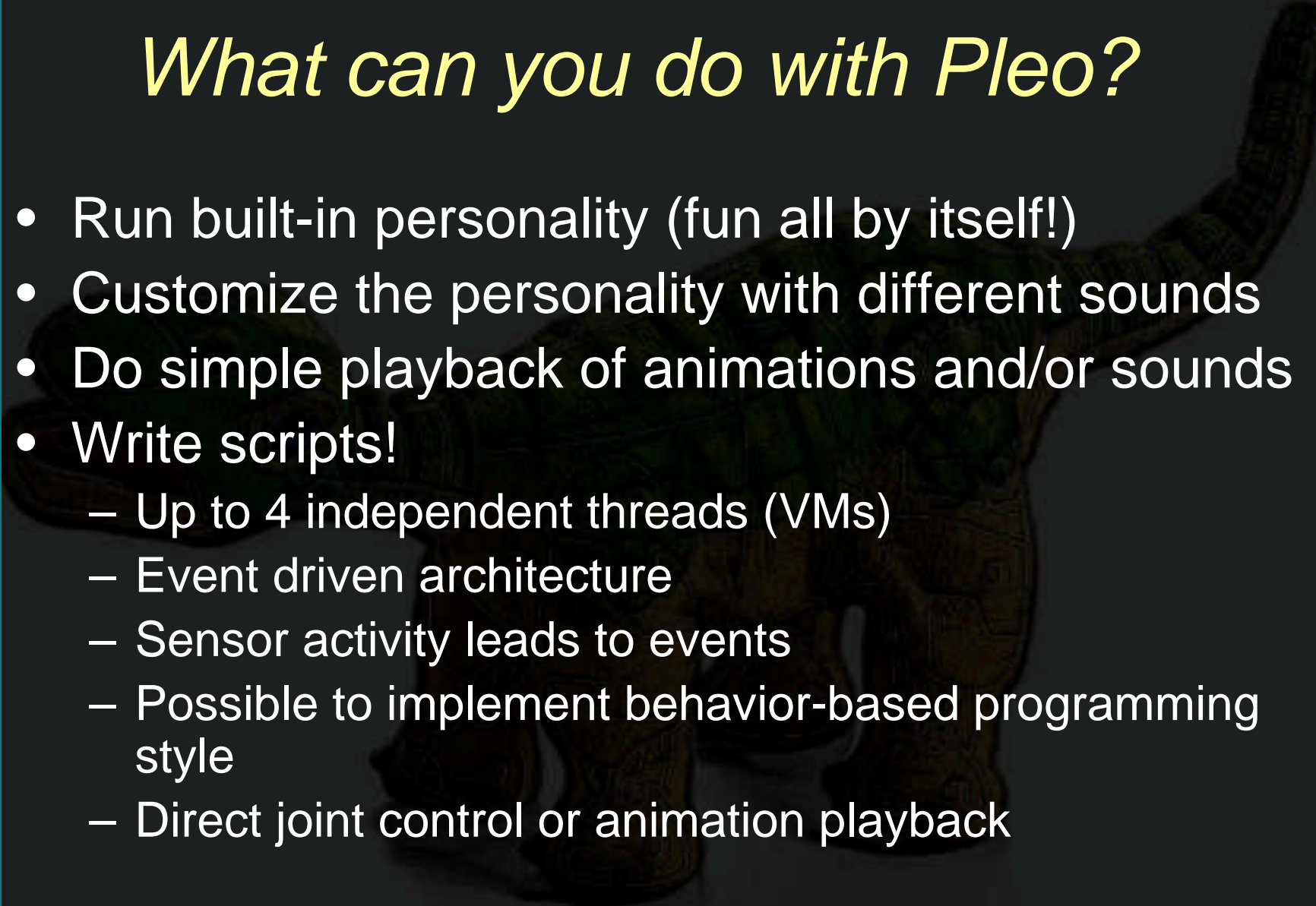
December 1, 2007

Pleo's LifeOS

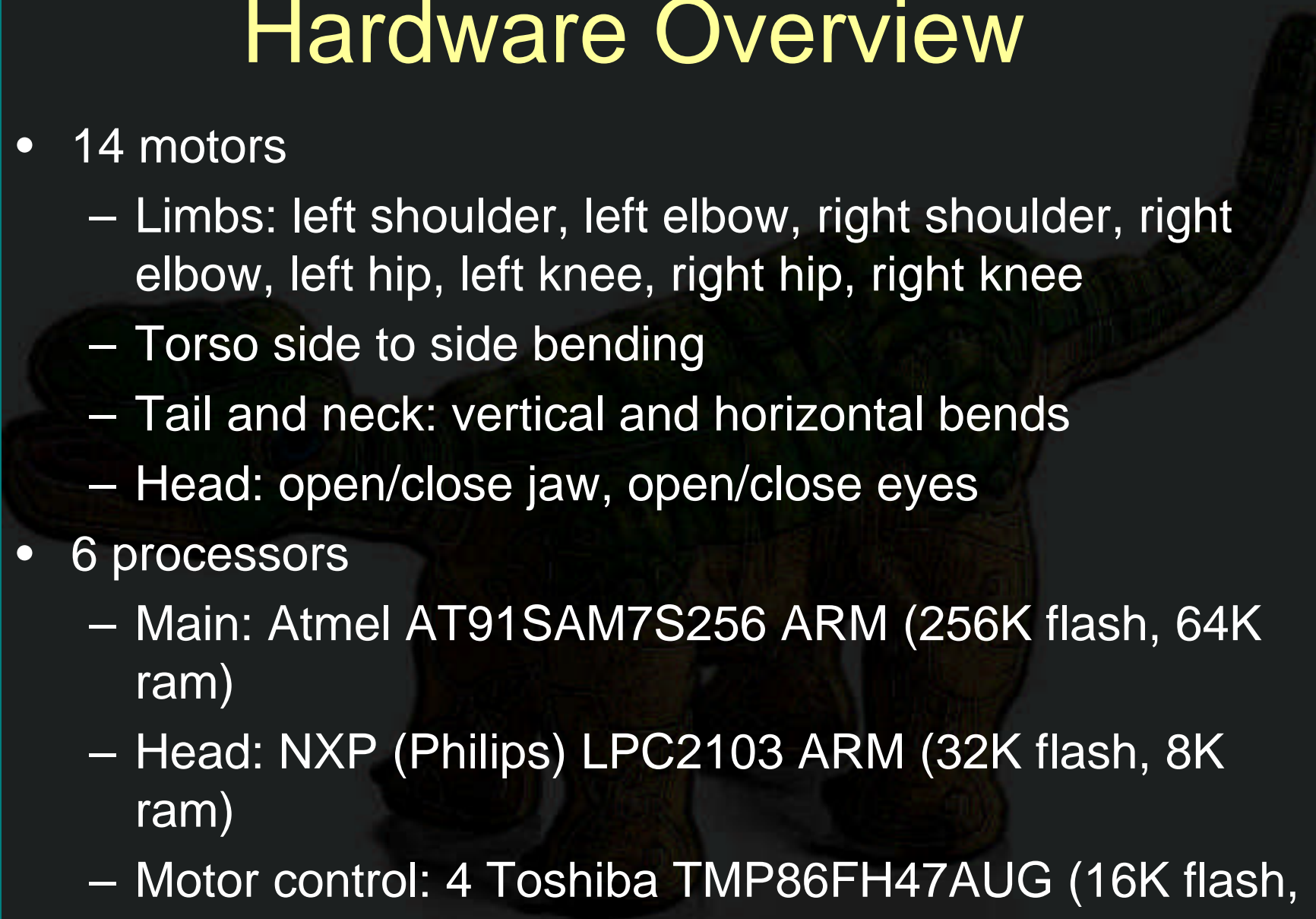
- Possibilities
 - Hardware Overview
 - Firmware Overview
 - Build Tools
 - Intro to Pawn
 - Execution Environment
 - API
 - Example Programs
- 

Possibilities:

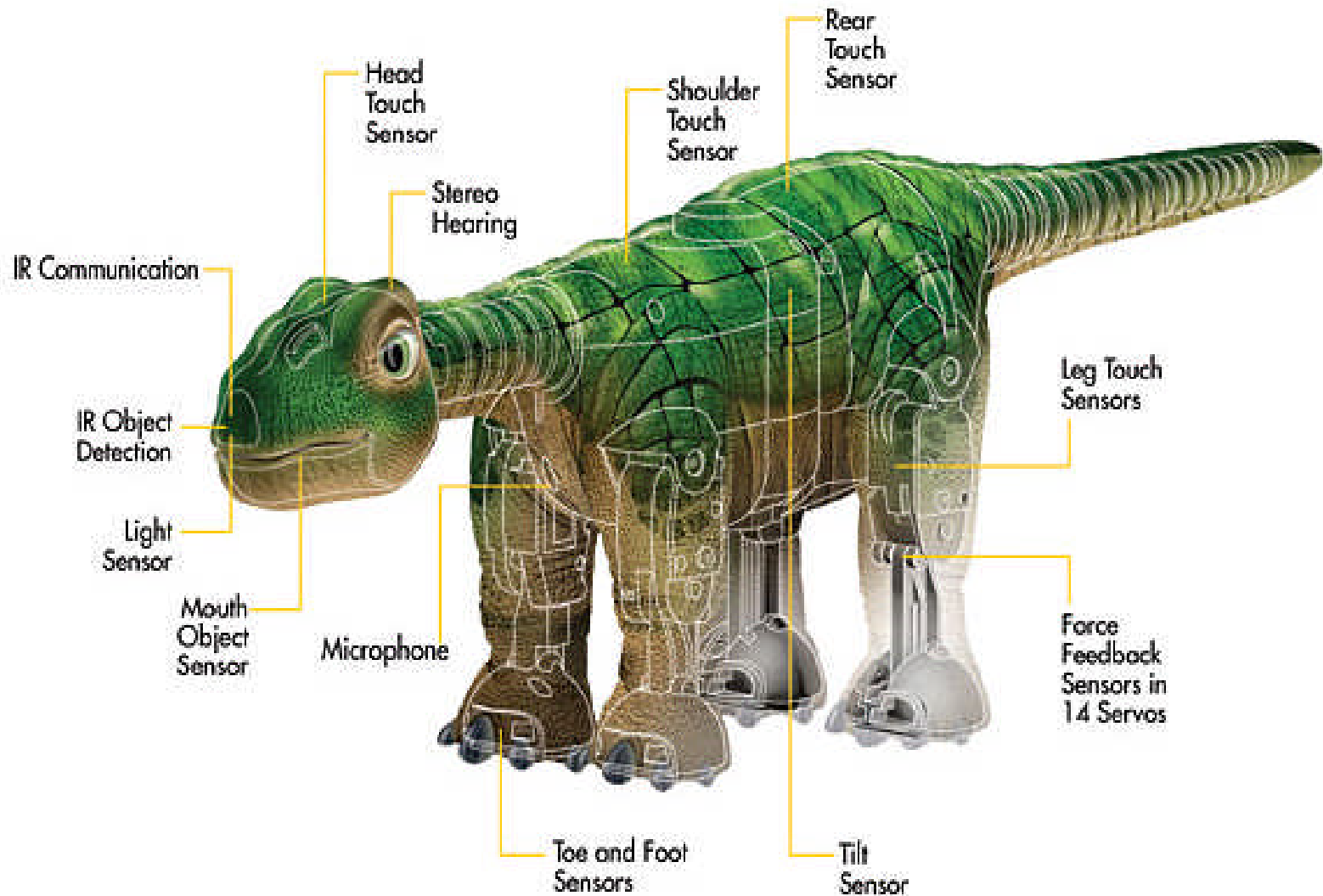
What can you do with Pleo?

- Run built-in personality (fun all by itself!)
 - Customize the personality with different sounds
 - Do simple playback of animations and/or sounds
 - Write scripts!
 - Up to 4 independent threads (VMs)
 - Event driven architecture
 - Sensor activity leads to events
 - Possible to implement behavior-based programming style
 - Direct joint control or animation playback
- 

Hardware Overview

- 14 motors
 - Limbs: left shoulder, left elbow, right shoulder, right elbow, left hip, left knee, right hip, right knee
 - Torso side to side bending
 - Tail and neck: vertical and horizontal bends
 - Head: open/close jaw, open/close eyes
 - 6 processors
 - Main: Atmel AT91SAM7S256 ARM (256K flash, 64K ram)
 - Head: NXP (Philips) LPC2103 ARM (32K flash, 8K ram)
 - Motor control: 4 Toshiba TMP86FH47AUG (16K flash, 512 ram)
- 

Hardware Overview...



Hardware Overview...

- 41 Sensors
 - 8 capacitive touch sensors (Quantum QT100)
 - 4 foot switches
 - 14 joint angle feedback potentiometers (VRs)
 - 2 microphones (sound direction and loudness)
 - 6 position orientation sensor
 - Shake sensor
 - IR obstacle sensor
 - Mouth sensor
 - Ambient light level (relative and absolute)
 - Color blob tracking
 - Battery current, temperature, and voltage

Firmware Overview

- Motor Control: PID position loops, trajectory planning (Toshiba)
- Head: Image/audio/IR sensing (NXP)
- Main: (Atmel)
 - Low Level: drivers, OS facilities – time keeping, file systems, logging, monitor interface, sensor data acquisition and processing, USB
 - Mid Level: sound, motion, property, script
 - High Level: personality or custom scripts

Pleo PDK Build Tools

- Prerequisites

- Python 2.4+ (to run tools on your computer)

- Required extensions: Element Tree (XML processing)

- Optional extensions: pyserial (direct comms to Pleo), PyGTK (Python bindings to GTK, for GUI tools)

- GTK (windowing toolkit for GUI tools)

- Pawn (to compile scripts to run in Pleo)

- <http://www.compuphase.com/pawn/pawn.htm>

- PDK tools (project builder, motion file converter, etc.)

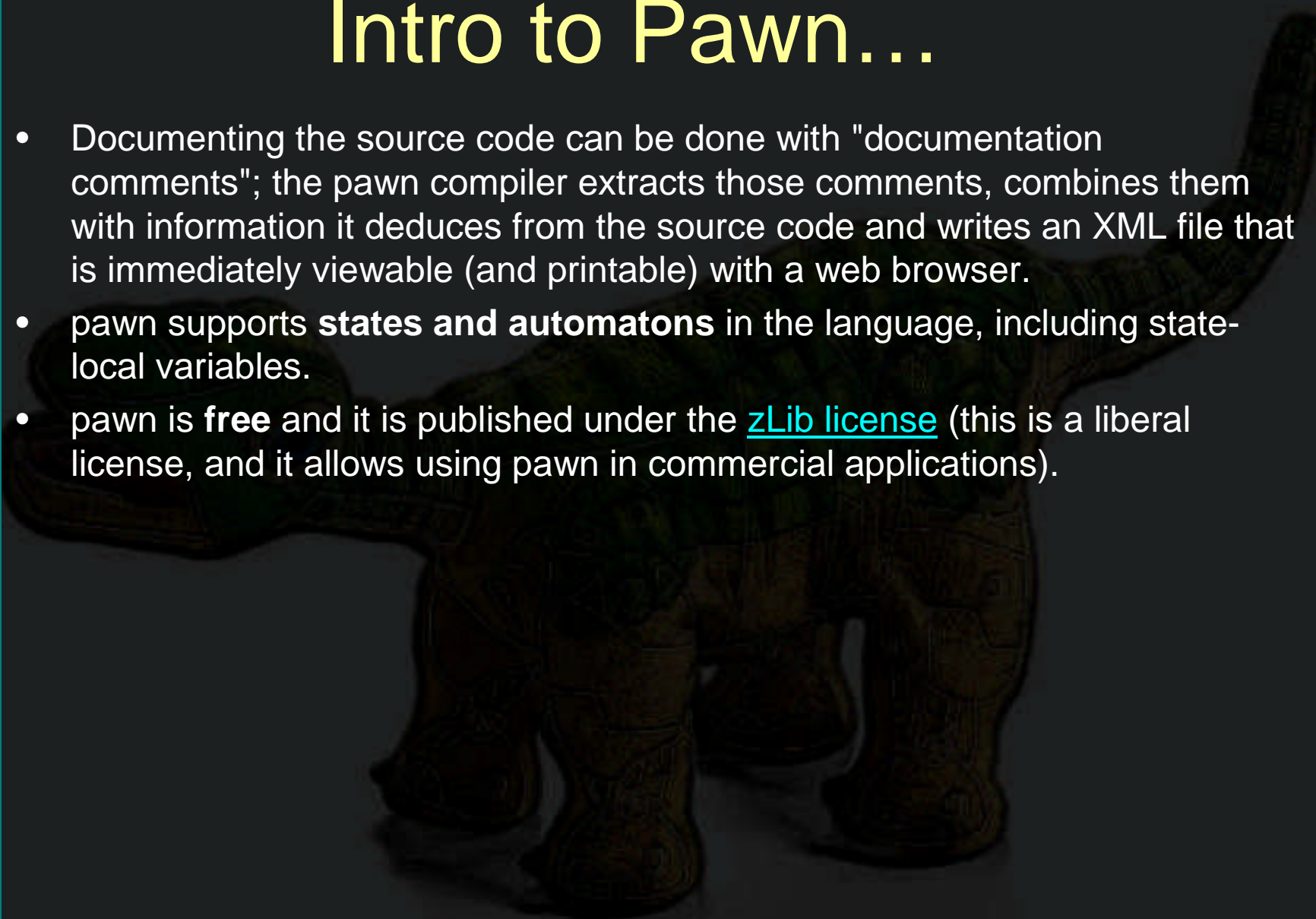
Intro to Pawn

- **An embedded scripting language formerly called Small**
- pawn is a simple, typeless, 32-bit extension language with a C-like syntax. A pawn "source" program is compiled to a binary file for optimal execution speed. The pawn compiler outputs P-code (or bytecode) that subsequently runs on an abstract machine. Execution speed, stability, simplicity and a small footprint were essential design criteria for both the language and the abstract machine.
- **Introduction**
- An introduction to the pawn language and abstract machine from a programmer's perspective was published in the October 1999 issue of [Dr. Dobb's Journal](#) --but at the time, the language was called Small. More verbose than the article, and more appropriate for non-expert programmers, is the manual. The manual contains a brief (tutorial) overview of the language, a language reference, programming notes on the abstract machine, casual notes about the why and how of many language features, and reference material.

Intro to Pawn...

- pawn is a simple, C-like, language.
- pawn is [a robust language](#) with a compiler that performs a maximum of static checks, and an abstract machine with (static) P-code verification and dynamic checks.
- For porting purposes, pawn is written in ANSI C as much as possible; Big Endian versus Little Endian is handled.
- pawn is *quick* (especially with Marc Peter's assembler implementation and/or his "just-in-time" compiler)
- pawn is small. It has been fitted on an Atmel ATmega128 microcontroller, Philips LPC2138 and LPC2106 microcontrollers (ARM7TDMI core with 32 kiB RAM), as well as on a Texas Instrument's MSP430F1611 (MSP430 core with 10 kiB RAM and 48 kiB Flash ROM).

Intro to Pawn...

- Documenting the source code can be done with "documentation comments"; the pawn compiler extracts those comments, combines them with information it deduces from the source code and writes an XML file that is immediately viewable (and printable) with a web browser.
 - pawn supports **states and automaton**s in the language, including state-local variables.
 - pawn is **free** and it is published under the [zLib license](#) (this is a liberal license, and it allows using pawn in commercial applications).
- 

Intro to Pawn...

“Why pawn now that there is Java, Lua, REXX, and countless others? Well, when I needed a language toolkit whose executable code can be embedded in resource files or animation file formats, that had a good interface to native functions, that added little overhead to the main application and could run on platforms and microcontrollers with (very) little RAM, and that was pretty fast, I could not really find an existing toolkit that fitted my needs.”

Thiadmer Riemersma, Compuphase

Inside the PDK

/SDK

```
  /bin
  ...tools needed to build and convert resources...
  /include
    File.inc
    Motion.inc
    ...other global includes...
    /common
      ...constants and types used across Ugobe products
  /pleo
    sensors.inc
    joints.inc
    ...other Pleo-specific includes...
  /projects
    /sensor_test
      /include
        ...local include files for this project...
      /build
        ...compiled files to be copied to SD Card...
        sensor_test.upf
        sensors.p
```

/media

```
  /motions
    ...motion csv files from animation software.
  /sounds
    ...sound wav files for sounds that Pleo can play.
  /commands
```

Simple Example

To start a new project:

- copy {pdk}/projects/template to new folder
- name it “sensor_test”
- Rename template.upf to sensors.upf (Ugobe Project File)

Directory structure:

```
/sensor_test
  /sounds
    beep.wav
  sensors.upf
  sensors.p
```

- Open sensors.p. Add this code to the on_sensor function:

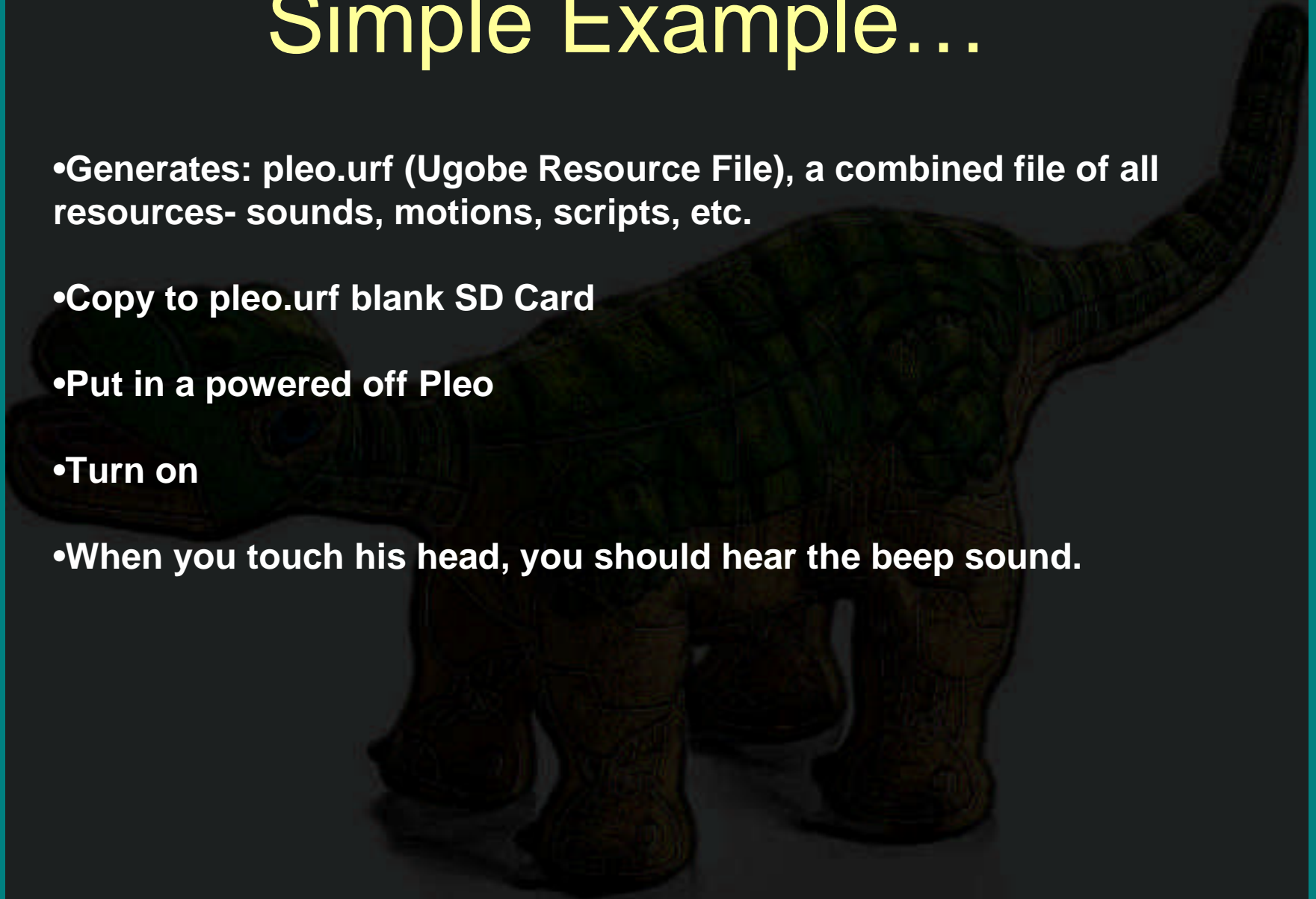
```
case SENSOR_HEAD:
    sound_play(snd_beep);
```

- Build the project:

```
python ../../bin/ugobe_project_tool.py sensor_test.upf rebuild
```

Simple Example...

- Generates: pleo.urf (Ugobe Resource File), a combined file of all resources- sounds, motions, scripts, etc.
- Copy to pleo.urf blank SD Card
- Put in a powered off Pleo
- Turn on
- When you touch his head, you should hear the beep sound.



LifeOS Scripts



Each script can contain these standard entry points:

```
public init() {  
}  
  
public main() {  
}  
  
public close() {  
}
```

The sensor script will also contain:

```
bool: on_sensor(time, sensor_name: sensor, value) {  
}
```

And can optionally contain:

```
on_property(time, property_name: property, value) {  
}
```

LifeOS API: VMs

Four VMs: Main, Sensor, Aux, User

Main VM: controlling script for the whole application; normally has a `main()`, which never returns; named **main.p -> main.amx**

Sensor VM: event handler: **sensors.p -> sensors.amx**:

bool: on_sensor(time, sensor_name: sensor, value);

where:

time is the time at which the sensor was triggered, given in milliseconds since Pleo was powered on

sensor is the ID of the sensor. See *sensor_name*.

value is the new value of this sensor after the trigger

Aux VM: for dynamically executed scripts (another VM switches with `exec()`); when the main exits, a property is set so the main script can call another; **anything.p -> anything.amx**

User VM: optional, but executed first if exists; can load / unload application (**init.p -> init.amx**)

LifeOS API: Include Files

Joint.inc: interface to control individual joints

Log.inc: interface to logging functions

Motion.inc: interface to control motion playback

Property.inc: interface to blackboard

Script.inc: interface to script VMs

Sensor.inc: interface to sensor system

Sound.inc: interface to control sound playback

Util.inc: interface to misc. utility functions: rand, time, etc.

LifeOS API: Properties

VMs have separate memory spaces; they can communicate with each other through **properties**. Properties have:

- Unique names
- Are integers
- Can be persisted to a file for retrieval later (e.g., after a power cycle)

Property API:

- bool property_load(const file_name[])
- bool property_save(const file_name[])

- int property_get(property_name: property)
- int property_set(property_name: property, value)

- void property_set_leak(property_name: property, delta, interval, max, nonlinear)

- property_enable_reporting(bool: enable)
- native bool: property_add_report(property_name: property, int: min_change, int: trigger)
- on_property(time, property_name: property, value)

LifeOS API: Joints

Joint IDs:

JOINT_RIGHT_SHOULDER, JOINT_RIGHT_ELBOW, JOINT_LEFT_SHOULDER, JOINT_LEFT_ELBOW,
JOINT_LEFT_HIP, JOINT_LEFT_KNEE, JOINT_RIGHT_HIP, JOINT_RIGHT_KNEE, JOINT_TORSO,
JOINT_TAIL_HORIZONTAL, JOINT_TAIL_VERTICAL, JOINT_NECK_HORIZONTAL,
JOINT_NECK_VERTICAL, JOINT_HEAD

Joint API:

Each joint has a range of possible angle values, which can be retrieved using the following functions:

- joint_get_min(joint_name: joint)
- joint_get_neutral(joint_name: joint)
- joint_get_max(joint_name: joint)

Joint movement functions include:

- void: joint_move_to(joint_name: joint, angle, speed, angle_type: type = angle_degrees);
- joint_get_position(joint_name: joint, angle_type);
- native bool: joint_is_moving(joint_name: joint);
- native void: joint_set_offset(joint_name: name, int: angle);
- native void: joint_control(joint_name: name, who);
(who = 1 means script is in control; 0 means motion file in control)

LifeOS API: Sensors

Sensor IDs:

SENSOR_BATTERY, SENSOR_IR, SENSOR_HEAD, SENSOR_CHIN, SENSOR_BACK,
SENSOR_LEFT_LEG, SENSOR_RIGHT_LEG, SENSOR_LEFT_ARM, SENSOR_RIGHT_ARM,
SENSOR_ARSE, SENSOR_TAIL, SENSOR_FRONT_LEFT, SENSOR_FRONT_RIGHT,
SENSOR_BACK_LEFT, SENSOR_BACK_RIGHT, SENSOR_CARD_DETECT, SENSOR_WRITE_PROTECT,
SENSOR_LIGHT, SENSOR_OBJECT, SENSOR_MOUTH, SENSOR_SOUND_DIR,
SENSOR_LIGHT_CHANGE, SENSOR_SOUND_LOUD, SENSOR_TILT, SENSOR_TERMINAL,
SENSOR_POWER_DETECT, SENSOR_USB_DETECT, SENSOR_WAKEUP, SENSOR_BATTERY_TEMP,
SENSOR_CHARGER_STATE, SENSOR_SHAKE, SENSOR_SOUND_LOUD_CHANGE

Sensor API:

- int sensor_get_value(sensor_name: sensor)
- int sensor_read_raw(sensor_name: sensor)
- bool sensor_is_triggered(sensor_name: sensor)
- sensor_reset(sensor_name: sensor)
- sensor_enable(sensor_name: sensor)
- sensor_disable(sensor_name: sensor)
- int: sensor_read_data(sensor_name: sensor, data[], length = sizeof data)
- int: sensor_data_size(sensor_name: sensor)
- sensor_set_config(sensor_name: sensor, sensor_config: type, int: value)
- native int: sensor_time_since_last(sensor_name: sensor)
- native int: sensor_get_trigger_time(sensor_name: sensor)
- native int: sensor_get_trigger_count(sensor_name: sensor)
- forward public on_sensor(time, sensor_name: sensor, value)

LifeOS API: Sound Playback

Pleo can play:

- Uncompressed
- 8 bit
- 1 channel
- 11025 Hz sample rate
- .wav files

Sound API:

- `sound_play(sound_name: sound, bool: interrupt = false);`
- `sound_play_file(const string[]);`
- `bool: sound_stop(Sound: sound = 0);`
- `bool: sound_is_playing(Sound: sound = 0);`
- `sound_set_volume(int: volume);`
- `sound_set_speed(int: speed);`

LifeOS API: Motion Playback

Pleo can play:

3dsmax (3D animation software) custom exporter outputs .csv files; PDK tool converts this to .umf file (Ugobe Motion File)

.csv files are frame-based, usually 30fps; position of all joints at each frame

.umf files are vector-based, curve-fit to each joint separately and stored in time-order of playback.

Motion API:

- Motion: motion_play(motion_name: name);
- Motion: motion_play_file(const file_name[]);
- bool: motion_pause(Motion: motion, bool: pause /*, bool: smooth = false*/);
- bool: motion_stop(Motion: motion);
- bool: motion_is_playing(Motion: motion);
- int: motion_set_playback_speed(int: percent_of_normal);

LifeOS API: Script

Scripts can control other scripts. They can launch another script, then block until that other script finishes, if desired. They can `yield()` to give up their timeslice.

They can wait for a property to reach a specified value (or inequality, including `>`, `<`, etc.)

Script API:

- int: `vm_exec(const string[], vm_name: vm = vm_aux);`
- Wait: `wait_new();`
- bool: `wait_add_property(Wait: wait, property_name: property, int: value, compare_name: compare);`
- void: `vm_wait(Wait: wait, vm_name: vm = vm_aux);`
- void: `yield();`

Example Programs: sensor test

```
//  
// Very basic sensor test: play a sound on each sensor  
//  
  
// Native functions  
#include <Sound.inc>  
#include <Sensor.inc>  
#include <Script.inc>  
#include <Log.inc>  
#include <Joint.inc>  
#include <Util.inc>  
  
// Local media  
#include "sounds.inc"  
forward play_beep(count);  
  
new sound_dir_time;  
new agc_on;  
new cur_angle;
```

Example Programs: Sensor Test...

```
// called once on entry
public init()
{
    new beacon_data[3];
    printf("Pleo Sensor Test Script: init\n");

    // enable our timer 'sensor'
    sensor_set_level(SENSOR_TIMER, 5000);
    sensor_enable(SENSOR_TIMER);
    beacon_data[0] = 0x5a;
    beacon_data[1] = 0xa5;
    beacon_data[2] = 0xbd;
    sensor_write_data(SENSOR_BEACON, beacon_data);
    sensor_read_data(SENSOR_BEACON, beacon_data, 1);
    printf("=== MY BEACON ID = %d ===\n", beacon_data[0])

    sound_dir_time = 0;
}

// called when we exit
public close()
{
    printf("close\n");
}
```

Example Programs: Sensor Test...

```
// called on each sensor trigger
public on_sensor(time, sensor_name:sensor, value)
{
    printf("on_sensor(%d,%d,%d)\n", sensor, value, time);
    switch (sensor)
    {
        // touch sensors
        case SENSOR_ARSE:
            if (value) sound_play(snd_sensor_arse, true)
        // foot switches
        case SENSOR_BACK_LEFT:
            if (value) sound_play(snd_sensor_back_left, true)
        case SENSOR_SHAKE:
            {
                printf("=== SENSOR_SHAKE: %d ===\n", value)
                if (value)
                    sound_play(snd_sensor_shake, true);
            }
        case SENSOR_LIGHT_CHANGE:
            {
                if (value > 10)
                    sound_play(snd_sensor_light_on, true);
                else if (value < -10)
                    sound_play(snd_sensor_light_off, true);
            }
    }
}
```

Example Programs: Sensor Test...

```
// send out some test IR data
case SENSOR_TIMER:
    sensor_write_data(SENSOR_IR, "Beep6789");
    // Receive IR data
    case SENSOR_IR:
    {
        new msg[48];
        sensor_read_data(SENSOR_IR, msg);
        printf("=== SENSOR_IR, msg received %s ===\n", msg);
    }
    case SENSOR_SOUND_DIR:
    {
        printf("on_sensor(%d,%d,%d)\n", sensor, value, time);
        if ((value != 0) && ((time - sound_dir_time) > 333))
        {
            if ((joint_get_attribute(JOINT_NECK_HORIZONTAL, ja_speed) == 0) &&
!sound_is_playing(0))
            {
                cur_angle = joint_get_position(JOINT_NECK_HORIZONTAL, angle_degrees)
                value = value / 2
                if (value > 0)
                    printf("sound to the right!\n");
                else if (value < 0)
                    printf("sound to the left!\n");
                else
                    printf("sound straight ahead!\n");
                printf("sound dir = %d; cur neck = %d; moving to %d\n",
                    value, cur_angle, value + cur_angle);
                joint_move_to(JOINT_NECK_HORIZONTAL, value + cur_angle);
                sound_dir_time = time;
            }
        }
    }
}
return true;
}
```

That's All Folks!