Brillo/Weave Part 2: Deep Dive

Open IoT Summit
Bruce Beare
April 2016
Acknowledgements...

Thank you to Google for the wonderful code labs and sample code.

Many of the examples and pictures are taken directly from the source tree, code labs or screen shots of Google tools.
Deep Dive into Brillo* and Weave*

- Code Structure (was covered in the Intro presentation)
- Hardware Access Layer
- Code Labs
- Monitoring and Analytics
- Logging
Brillo* / Android* - HAL Structure

HALs are:

- Defined by Google* header files
- Common with Android (mostly)
- Not all implemented or required (yet)

```
activity_recognition.h  bt_rc.h
audio_alsaops.h        bt_sdpg.h
audio_effect.h         bt_sock.h
audio.h                camera2.h
audio_policy.h         camera3.h
bluetooth.h            camera_common.h
boot_control.h         camera.h
bt_av.h                consumerir.h
bt_common_types.h      fb.h
bt_gatt_client.h       fingerprint.h
bt_gatt.h              fused_location.h
bt_gatt_types.h        gatekeeper.h
bt_hf_client.h         gps.h
bt_hf.h                gralloc.h
bt_hh.h                hardware.h
bt_hl.h                hdmi_cec.h
bt_mce.h               hw_auth_token.h
bt_pan.h               hwcomposer_defs.h

input.h                keymaster0.h
bt_sdpg.h              keymaster1.h
bt_sock.h              keymaster_common.h
bt_sdpg.h              keymaster_defs.h
lights.h               local_time_hal.h
memtrack.h             nfc.h
bt_sdpg.h              nfc_tag.h
input.h                power.h
bt_sdpg.h              qemu.h
bt_sdpg.h              qemu_pipe.h
bt_sdpg.h              radio.h
bt_sdpg.h              sensors.h
hwcomposer_defs.h      sound_trigger.h
hwcomposer.h           tv_input.h
vibrator.h             
```
WiFi HAL

- Largely handled by wpa_supplicant
- Requires NL802.11 support
- Responsible for initialization
- Performs station/AP mode switching
- Brillo* specific HAL

```c
typedef enum {
   WIFI_MODE_STATION = 0,
   WIFI_MODE_AP = 1
} wifi_driver_mode;

typedef struct wifi_driver_device {
   hw_device_t common;
   /**<
      * Initialize the driver. Network devices may or may not appear as
      * a result.
      *
      * @return WIFI_SUCCESS if successful, or appropriate wifi_driver_error.
      */
   wifi_driver_error (*wifi_driver_initialize)();
   /**<
      * Start up the device in a specified mode. It is guaranteed that the
      * wifi_driver_initialize function has been called prior to this. This
      * function can be called multiple times to switch between modes.
      *
      * @param mode mode to start the driver up in.
      * @param wifi_device_name_buf name of the device created (on success).
      * @param wifi_device_name_size number of bytes available in device_name.
      *
      * @return WIFI_SUCCESS if successful, or appropriate wifi_driver_error.
      */
   wifi_driver_error (*wifi_driver_set_mode)(wifi_driver_mode mode,
                                           char* wifi_device_name_buf,
                                           size_t wifi_device_name_size);
} wifi_driver_device_t;
```
Low-Speed Bus Interface Libraries

**libupm**
- High level implementations for many sensors
- C++ Class bindings
- Easy to use, modify, extend

**libmraa**
- I/O and bus access library
- Architecture Independant
- Supports many boards
- Bindings for C++, Python*, Java*, JavaScript

### C API Modules vs C++ API Classes

<table>
<thead>
<tr>
<th>C API Modules</th>
<th>C++ API Classes</th>
</tr>
</thead>
<tbody>
<tr>
<td>gpio</td>
<td>Gpio class</td>
</tr>
<tr>
<td>i2c</td>
<td>I2c class</td>
</tr>
<tr>
<td>aio</td>
<td>Aio class</td>
</tr>
<tr>
<td>pwm</td>
<td>Pwm class</td>
</tr>
<tr>
<td>spi</td>
<td>Spi class</td>
</tr>
<tr>
<td>uart</td>
<td>Uart class</td>
</tr>
<tr>
<td>common</td>
<td>common</td>
</tr>
</tbody>
</table>

### Supported platforms

- Intel® Galileo Gen 1 - Rev D
- Intel® Galileo Gen 2 - Rev H
- Intel® Edison kit
- Intel® NUC DE3815tykhe
- Intel® Expansion Kit with Minnowboard MAX*
- Raspberry Pi*
- Banana Pi*/Pro
- Beaglebone Black*
- Intel® NUC NUC5i5MYBE
- FTDI FT4222H
HAL Structure with libmraa, libupm

Application Daemons:

- Receive getter/setter requests via Weave [binder as the internal transport]
- Are MAC access controlled via SELinux
- Instantiate libupm classes to implement a particular sensor or device
- May call directly into libmraa for direct access to pins and busses
- May make system calls directly for access to non-mraa devices
Sensor Service and HAL

A single process talks to the sensor HAL

- Race and bus issues are handled within the HAL (libmraa and libupm can mutex if required)
- The HAL can talk either via libmraa, libupm... or directly to the kernel

- The HAL upper layer is defined by: hardware/libhardware/include/hardware/*.h
- The HAL lower layer is implemented by: sensors.<device>.so and any code that is called by the dynamic library.
Intel’s Sensor HAL

- The i2c bus is auto-detected by libmraa
- The main thread sleeps if there is no event available (power consumption friendly)
- Each activated sensor has a thread which
  - periodically reads data from the sensor
  - writes it to a pipe connected to the main thread

Adding a sensor is very simple
- Just add a new file with the new Sensor class and include it in Android.mk.
- Selection of the supported sensors can be done at build time
Peripheral Manager

- Arbitrate access to busses and pin multiplexers across processes
- SELinux security model for bus/device access
- Increase re-use of peripheral HALs
- Utilize Pin names for functions rather than board specific numbers
- Loadable (pluggable) HALs for specific devices
- ABI Stability, consistency and future sharing with Android

The Peripheral manager is likely to replace libmraa ... stay tuned
Other HALs in use by Brillo*

Audio HAL
- Android* Audio HAL, we support USB, I2S audio

Bluetooth* HAL
- Able to scan and advertise... waiting for Google* to complete the BT framework.

Lights & Notification HAL
- Android Lights/notification, generally just turn on/off a GPIO

Storage HAL
- Direct access to the mounted file systems via C, C++
# Calling a HAL directly

```c
#include <err.h>
#include <stdio.h>
#include <unistd.h>
#include <hardware/hardware.h>
#include <hardware/lights.h>

int main() {
    const hw_module_t* module = nullptr;
    struct light_device_t* light_device = nullptr;

    int ret = hw_get_module(LIGHTS_HARDWARE_MODULE_ID, &module);
    if (ret || !module)
        err(1, "Failed to load %s", LIGHTS_HARDWARE_MODULE_ID);

    ret = module->methods->open(
        module, LIGHT_ID_NOTIFICATIONS,
        reinterpret_cast<struct hw_device_t**>(&light_device));
    if (ret || !light_device)
        err(1, "Failed to open %s", LIGHT_ID_NOTIFICATIONS);

    struct light_state_t state = {
        .color = 0,
        .flashMode = LIGHT_FLASH_NONE,
        .flashOnMS = 0,
        .flashOffMS = 0,
        .brightnessMode = 0,
    };

    // Turn light on for three seconds.
    state.color = 1;
    light_device->set_light(light_device, &state);
    sleep(3);

    // Flash for three seconds.
    state.flashMode = LIGHT_FLASH_TIMED;
    state.flashOnMS = 50;
    state.flashOffMS = 50;
    light_device->set_light(light_device, &state);
    sleep(3);

    // Turn light off.
    state.color = 0;
    state.flashMode = LIGHT_FLASH_NONE;
    light_device->set_light(light_device, &state);

    light_device->common.close(
        reinterpret_cast<struct hw_device_t*>(light_device));

    return 0;
}
```
Google* Code Labs

Step by Step Learning for Brillo* and Weave*

- Hello World for Brillo IO Programming
- Hello World Weave Integration
- Enable Google Services and APIs
- Web Service Development

http://developers.google.com/brillo
Brillo* Hello World (with libmraa)
https://codelabs.developers.google.com/codelabs/brillo-hello-leds-edison

• Download the BDK
• brunch: Add the BSP, build/flash the image
• Build the LED app program
• Flash the device
• Execute.... and watch the blinky lights
$ TOP=/build
$ export BDK_PATH=$TOP/bdk
$ cd $TOP

$ curl https://dl.google.com/dl/brillo/bdk/latest/bdk-latest.tar.gz
$ tar xf bdk-latest.tar.gz

$ cd $TOP/bdk
$ tools/bdk/brunch/brunch bsp download edison

$ mkdir $TOP/products && cd $TOP/products
$ ${BDK_PATH}/tools/bdk/brunch/brunch product create helloLEDs edison

$ cd $TOP/products/helloLEDs
$ echo hello_led_service >> config/packages

$ cd $TOP/products/helloLEDs/src
$ mkdir hello_led_service
#include <unistd.h>
#include <stdio.h>
#include <mraa.h>

int main(__unused int argc, __unused char* argv[]) {
    const unsigned gpio[] = {0, 1, 2};
    const int gpio_count = sizeof(gpio)/sizeof(*gpio);
    mraa_gpio_context m_gpio[] = {NULL, NULL, NULL};

    mraa_init();

    for (int i = 0; i < gpio_count; i++) {
        m_gpio[i] = mraa_gpio_init(gpio[i]);
        if (!m_gpio[i])
            return 1;
        mraa_gpio_dir(m_gpio[i], MRAA_GPIO_OUT);
    }

    for (int iterations=0; iterations < 3; iterations++) {
        for (int i = 0; i < gpio_count; i++) {
            mraa_gpio_write(m_gpio[i], 1);
            sleep(1);
        }
        for (int i = gpio_count-1; i >= 0; i--)
            mraa_gpio_write(m_gpio[i], 0);
            sleep(1);
    }

    for (int i = 0; i < gpio_count; i++)
        mraa_gpio_close(m_gpio[i]);
    return 0;
}
Write your App, Build, Flash

hello_led_service/Android.mk

LOCAL_PATH := $(call my-dir)
include $(CLEAR_VARS)
LOCAL_MODULE := hello_led_service
LOCAL_SRC_FILES := hello_led_service.cpp
LOCAL_SHARED_LIBRARIES := libc libbase libmraa
LOCAL_CFLAGS := -Werror
include $(BUILD_EXECUTABLE)

$ cd ~/products/helloLEDs
$ source envsetup.sh
$ m -j 20
$ provision
$ fastboot reboot
$ adb shell hello_led_service
$ du -s -h .
13G .
$
$ cd src/helloLed_service/
$ touch helloLed_service.cpp
$ mm -j20
Building submodule only . . .
Output will be placed in /spare/PROD/helloLEDs/out
...
[ 14% 1/7] target C++: helloLed_service <= /spare/PROD/helloLEDs/src/helloLed_service/helloLed_service.cpp
...
### make completed successfully (1 seconds) ###

$ adb root
restarting adbd as root

$ adb remount
remount succeeded

$ adb sync
/system/: 1 file pushed. 706 files skipped. 0.0 MB/s (5516 bytes in 0.225s)K
/data/: 0 files pushed. 7 files skipped.

$ adb shell helloLed_service
$
Google Code Labs

- Hello World for Brillo IO Programming
- Hello World Weave Integration
- Enable Google Services and APIs
- Web Service Development
Binder integration with C++

Binder now has a fully supported C++ interface

- aidl compiler at: out/host/linux-x86/bin/aidl-cpp
- Generates a *.cpp file
- Compiles to a *.a and linked into the program
- Interface defined in:
  frameworks/native/include/binder/IInterface.h

```cpp
#define SERVICE "brillo.examples.ledflasher.ILEDService"
namespace brillo {
  namespace examples {
    namespace ledflasher {
      IMPLEMENT_META_INTERFACE(LEDService, SERVICE);
    }  // namespace ledflasher
  }  // namespace examples
}  // namespace brillo
```

```make
common/Android.mk:
  LOCAL_PATH := $(call my-dir)
  include $(CLEAR_VARS)
  LOCAL_MODULE := libledservice-common
  LOCAL_AIDL_INCLUDES := $(LOCAL_PATH)/aidl
  LOCAL_EXPORT_C_INCLUDE_DIRS := $(LOCAL_PATH)
  LOCAL_SRC_FILES :=
    aidl/brillo/examples/ledflasher/ILEDService.aidl
    binder_constants.cpp
  include $(BUILD_STATIC_LIBRARY)
```

```aidl
common/aidl/brillo/examples/ledflasher/ILEDService.aidl:
  package brillo.examples.ledflasher;
  interface ILEDService {
    void setLED(int ledIndex, boolean on);
    boolean getLED(int ledIndex);
    boolean[] getAllLEDs();
  }
```

```cpp
common/binder_constants.h:
  namespace ledservice {
    extern const char kBinderServiceName[];
  }
```

```cpp
common/binder_constants.cpp:
  #include "binder_constants.h"
  namespace ledservice {
    const char kBinderServiceName[] = "example_led_service";
  }
```
Weave* Schema

Weave requests are:

• sent from the web via XMPP over any available network transport
• forwarded to the Brillo* application server via the Binder
• Commands or states associations are defined in a .json file installed on the target

A Weave app (web, IOS, Android) introspects the device capabilities and can then construct an appropriate interface

```json
$ cat ledflasher.json
{
  "_ledflasher": {
    "commands": {
      "set": {
        "minimalRole": "user",
        "parameters": {
          "led": {
            "type": "integer",
            "minimum": 1,
            "maximum": 4
          }
        }
      },
      "on": {
        "type": "boolean"
      }
    }
  },
  "toggle": {
    "minimalRole": "user",
    "parameters": {
      "led": {
        "type": "integer",
        "minimum": 1,
        "maximum": 4
      }
    }
  }
}
```
Binding Weave* Handlers

New Coding Constructs:
- Google* has imported the use of WeakPtr from Chromium*
- Google has imported the use of TaskRunner from Chromium

https://www.chromium.org/developers/coding-style/important-abstractions-and-data-structures

https://www.chromium.org/developers/design-documents/threading

```cpp
#define WEAK_PTR_THIS weak_ptr_factory_.GetWeakPtr()

void Daemon::OnWeaveServiceConnected(
    const std::weak_ptr<weaved::Service>& service) {
  weave_service_ = service;
  auto weave_service = weave_service_.lock();
  if (!weave_service)
    return;

  weave_service->AddComponent(kWeaveComponent,
                               {kWeaveTrait}, nullptr);

  weave_service->AddCommandHandler(
      kWeaveComponent, kWeaveTrait, "set",
      base::Bind(&Daemon::OnSet, WEAK_PTR_THIS));

  weave_service->AddCommandHandler(
      kWeaveComponent, kWeaveTrait, "toggle",
      base::Bind(&Daemon::OnToggle, WEAK_PTR_THIS));
  ...

  weave_service->SetPairingInfoListener(
      base::Bind(&Daemon::OnPairingInfoChanged,
                 WEAK_PTR_THIS));

  UpdateDeviceState();
}
```
Weave* Service Handler

- Invoked when a weave message is received via the Binder
- Executed in a TaskRunner context

```cpp
void Daemon::OnSet(std::unique_ptr<weaved::Command> command) {
    if (!led_service_.get()) {
        command->Abort("_system_error", "ledservice", nullptr);
        return;
    }

    int index = command->GetParameter<int>("led");
    if (index < 1 || index > 4) {
        command->Abort("_invalid_parameter", "Invalid", nullptr);
        return;
    }

    bool on = command->GetParameter<bool>("on");
    android::binder::Status status =
        led_service_ ->setLED(index - 1, on);

    if (!status.isOk()) {
        command->AbortWithCustomError(status, nullptr);
        return;
    }

    animation_.reset();
    status_ = "idle";

    UpdateDeviceState();
    command->Complete({}, nullptr);
}
```
Weave* Device State

Sent to the web via Weave whenever device the state changes

```
void Daemon::UpdateDeviceState() {
  if (!led_service_.get())
    return;
  std::vector<bool> leds;
  if (!led_service_->getAllLEDs(&leds).isOk())
    return;
  auto weave_service = weave_service_.lock();
  if (!weave_service)
    return;
  brillo::VariantDictionary state_change{
    {"_ledflasher.status", status_},
    {"_ledflasher.leds", leds},
  };
  weave_service->SetStateProperties
    (kWeaveComponent, state_change, nullptr);
}
```

.../ledflasher/ledflasher.json:

```
{
  "_ledflasher": {
    ...
    "state": {
      "status": {
        "type": "string",
        "enum": ["idle", "animating"]
      },
      "leds": {
        "type": "array",
        "items": {"type": "boolean"}
      }
    }
  }
}
```
**Build, Flash, Test**

- Incremental builds are much quicker.
- USE_CCACHE=1 can speed rebuilds
- Either re-provision (flash) the device or push the files out to the device

You are now ready to register your device with the cloud.

```bash
$ source envsetup.sh
Brillo Development Kit v.10.0.0 from /spare/AOSP
Available commands:
brunch, m, mm, adb, fastboot, provision, gdbclient.py
```

```bash
$ m -j20 USE_CCACHE=1
Output will be placed in /spare/PROD/ledflasher/out...
...[ 97% 35/36] build /spare/PROD/ledflasher/out/out-edison/target/product/edison/kernel
[100% 36/36] Target boot image: /spare/PROD/ledflasher/out/out-edison/target/product/edison/boot.img
### make completed successfully (03:18 (mm:ss)) ###
```

```bash
$ adb root
adb is already running as root

$ adb remount
```

```bash
$ adb sync
/system/: 5 files pushed. 702 files skipped. 1.1 MB/s (175755 bytes)
data/: 0 files pushed. 7 files skipped.
```

```bash
$ adb reboot
$```
Registering (provisioning) a Brillo* Device

https://codelabs.developers.google.com/codelabs/brillo-weave-leds

- Register (provision) your device from the Chrome Browser
Android* Weave Application

https://play.google.com/apps/testing/com.google.android.weave

• Register (provision) your device from Android
Sending Weave* Commands
Viewing Weave* Device State
• Hello World for Brillo IO Programming
• Hello World Weave Integration
• Enable Google Services and APIs
• Web Service Development

What you’ll learn
✓ Registering new product in the Weave Developers Console.
✓ How to enable Weave for your product by creating credentials and adding a model manifest.
✓ Copying credentials and identifiers to your build environment.
✓ Sending commands in the Weave Developers Console.

What you’ll need if you’re building with Weave on Brillo (recommended for this Codelab and learning Weave)
• A Brillo compatible development board flashed with Brillo (if you’re building on Brillo)
• A Linux workstation with at least ~200GB free disk space.
• The latest BDK installed on your workstation (instructions)
• A USB micro to USB data cable.
• An Android device or the Google Chrome Browser on Win/Mac/Chromebook.
• Successful completion of the Hello World codelab (Edison, Dragonboard, others)
• (Recommended: Successful completion of the Weave Integration codelab as well.)

Note: As in the previous codelabs, these instructions assume your base Brillo directory is ${BDK_PATH} and your product directory is ~/products/.

What you’ll need if you’re building with Weave on a non-Brillo Linux OS
• An Android device or the Google Chrome Browser on Win/Mac/Chromebook
• (A sample) wrapper implementation of libweave as a daemon on Linux, see Weave’s Get Started (this, essentially...
Google Cloud and Developer’s Consoles

https://console.cloud.google.com
https://console.developers.google.com

- Create Google Cloud Projects
- Enable API use
- Manage App Engine
- Manage Compute Engine
- Documentation
Weave* Developer’s Console
https://weave.google.com/console
Create a New Product (Cloud Project)

https://weave.google.com/console -> Your products
Google* Credentials Console

https://console.cloud.google.com/apis/credentials

- Your device keys were automatically created for your project.
- Create your Browser and Web keys if you plan to write a web client
Copy Identifiers and Keys to your Build Environment

### weaved.conf

```conf
# OAuth 2.0 client id.
client_id=247106653574-ki66j1eI06dbhvpdk0098t28fteneke.apps.googleusercontent.com

# OAuth 2.0 client secret.
client_secret=Mx8NSPEPdnAOHqYGzEByYdOt

# API key.
api_key=AIzaSyCwqT2dpFFezXzLcrqaFqiRYdUMucUz2jU

# Human readable name of the device.
name=Lights on Edison

# Human readable description of the device.
# Defaults to empty string"
description=Blink lights on an Edison Development Board

# Manufacturer of the device.
oem_name=FSMco

# Model of the device.
model_name=StarryNight

# Five character code assigned by the cloud registry
model_id=ABvGf
```

<table>
<thead>
<tr>
<th>Model ID</th>
<th>Device kind</th>
<th>Model name (required)</th>
<th>OEM name (required)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABvGf</td>
<td>developmentBoard</td>
<td>Lights on Edison</td>
<td>FSMco</td>
</tr>
</tbody>
</table>
Google* Code Labs

- Hello World for Brillo IO Programming
- Hello World Weave Integration
- Enable Google Services and APIs
- Web Service Development
Web API Bindings

Implement Web Apps with a variety of programming languages:
• Shell (curl)
• Python
• Java
• JavaScript

Use https: to execute Google Cloud APIs.
Prototyping Weave commands with CURL

RESTful APIs
• Uniform Interface
• Stateless
• Cacheable
• Client-Server
• Layered System
• Code on Demand (optional)

JSON Data Format

OAuth Token for authentication

Prototype and test commands with CURL
• List all devices
• Get device information
• Insert (send) a Command to the device

https://developers.google.com/weave/guides/web-dev/web-service-dev
Sending RESTful commands via CURL

Device Commands:
• Delete
• Get
• handleInvitation
• List
• Patch
• patchState
• Update
• updateParent
• upsertLocalAuthInfo

https://developers.google.com/weave/v1/reference/cloud-api/devices
#!/bin/bash
# file "functions"

# Gets the client_id, client_secret from weaved.conf
eval $(egrep '^client_id='|'^client_secret=' ../weaved.conf)

# Run authorize_account.sh to get the following token
refresh_token="1/sYKiBDwPbFO_PM1GI-UBYW1D01jw09hqlui-AcVGdttU"

# Refresh the auth_token from the refresh token
# return the new auth_token
function auth_token {
    curl -s -d "
    refresh_token=$refresh_token&
    client_id=$client_id&
    client_secret=$client_secret&
    grant_type=refresh_token"
    https://accounts.google.com/o/oauth2/token |
    jq $.access_token
}

# Return the device id for the first found device
# function device_id {
#    curl -s -H "Authorization: Bearer $(auth_token)"
#    https://www.googleapis.com/clouddevices/v1/devices |
#    jq $.devices[0].id
#
#} # return the .json for the command definitions
# function command_defs {
#    curl -s -H "Authorization: Bearer $(auth_token)"
#    https://www.googleapis.com/clouddevices/v1/devices |
#    jq $.devices[0].commandDefs'
}
Authorize Account and get a Refresh Token

```bash
#!/bin/bash
source functions

google-chrome "https://accounts.google.com/o/oauth2/auth?"  
scope=https://www.googleapis.com/auth/clouddevices&
redirect_uri=urn:ietf:wg:oauth:2.0:oob&
response_type=code&
client_id=$client_id"

echo -n "Enter the auth_code from the browser window: "
read auth_code

curl -d "
  code=$auth_code&
  client_id=$client_id&
  client_secret=$client_secret&
  redirect_uri=urn:ietf:wg:oauth:2.0:oob&
  grant_type=authorization_code"  https://accounts.google.com/o/oauth2/token

echo

echo "Copy the above refresh_token to the 'functions' file."```
List All Registered Devices

```
#!/bin/bash
source ./functions

curl -s -H "Authorization: Bearer $(auth_token -r)" \
  https://www.googleapis.com/clouddevices/v1/devices |  
  jq '.devices[].id'
```

- "52c867ca-17d5-f422-a2c8-b31c4e02743e"
- "e28ea71b-8115-cd76-4852-b08f9bd6faf0"
- "3637ff18-2538-0973-fe01-d45d8a91e676"
- "ed61c2fc-f7c9-0936-ce23-a498b1d9a3ad"
- "64d6336a-5bbac971-4f51-bb550a4a7529"
- "be8edca8-8e1b-c480-8595-c9cc335ec8d4"
- "3242af3d-eb28-8ca8-0f20-4dcd2d112624"
- "fd2437c0-f421-4ea2-05f9-4199e5c1ab2b"

https://developers.google.com/weave/v1/reference/cloud-api/devices/list
Get State for a Specific Device

```bash
#!/bin/bash
source ./functions


{}

  "leds": [
    false,
    false,
    false,
    false
  ],
  "status": "idle"
}
List available device commands

```bash
#!/bin/bash
source ./functions

curl -H "Authorization: Bearer $(auth_token -r)"
  "https://www.googleapis.com/weave/v1/commands?deviceId=$(device_id)

{
  "kind": "weave#command",
  "id": "52c867ca-17d5-f422-a2c8-b31c4e02743e5d194dbc-dbc8-f4c2-16f2-6f01a9dd228b",
  "deviceId": "52c867ca-17d5-f422-a2c8-b31c4e02743e",
  "creatorEmail": "bbeare1.test@gmail.com",
  "component": "base",
  "name": "base.updateDeviceInfo",
  "parameters": {
    "location": "Home office"
  },
  "state": "done",
  "creationTimeMs": "1457726873129",
  "expirationTimeMs": "1457726883129",
  "expirationTimeoutMs": "10000"
},
{
  "kind": "weave#command",
  "id": "52c867ca-17d5-f422-a2c8-b31c4e02743e5d194dbc-dbc8-f4c2-16f2-6f01a9dd228b",
  "deviceId": "52c867ca-17d5-f422-a2c8-b31c4e02743e",
  "creatorEmail": "bbeare1.test@gmail.com",
  "component": "base",
  "name": "base.updateDeviceInfo",
  "parameters": {
    "name": "Bruce: Lights on Edison"
  },
  "state": "done",
  "creationTimeMs": "1457726873129",
  "expirationTimeMs": "1457726883129",
  "expirationTimeoutMs": "10000"
}
"totalResults": 2
}
```

https://developers.google.com/weave/v1/reference/cloud-api/devices/list
Turn ON/OFF an LED

#!/bin/bash
source ./functions

body="{
   'deviceId': $(device_id),
   'name': '_ledflasher.set',
   'parameters': {
      'led': 1,
      'on': true
   }
}"

curl -H "Authorization: Bearer $(auth_token)"
   -H "Content-Type: application/json" -X POST
   -d "$body"
   https://www.googleapis.com/clouddevices/v1/commands
Google Code Labs

Additional Topics

- Over the Air Updates
- IOS/Android Application Development
Questions?
Intel technologies’ features and benefits depend on system configuration and may require enabled hardware, software or service activation. Learn more at intel.com, or from the OEM or retailer.

No computer system can be absolutely secure.

Tests document performance of components on a particular test, in specific systems. Differences in hardware, software, or configuration will affect actual performance. Consult other sources of information to evaluate performance as you consider your purchase. For more complete information about performance and benchmark results, visit http://www.intel.com/performance.

Intel, the Intel logo and others are trademarks of Intel Corporation in the U.S. and/or other countries. *Other names and brands may be claimed as the property of others.

© 2016 Intel Corporation.