

Capítulo 4.

Android Things: Entradas / Salidas

Por JESÚS TOMÁS

El término “Internet de las Cosas” es sin duda una de las palabras de moda dentro y fuera de los círculos tecnológicos. Como suele pasar en estos casos, se ha extendido tanto y en tan poco tiempo que ha dado pie a multitud de interpretaciones y definiciones sobre cuáles son exactamente sus capacidades, o sobre donde empieza y donde termina exactamente un proyecto de Internet de las Cosas. Sin embargo, su propio nombre es tan autodescriptivo que no deja lugar a dudas de cuáles son sus pilares fundamentales: **las Cosas**, esa interacción con el mundo físico, una digitalización mediante soluciones electrónicas y de sistemas embebidos que permite transformarlas al mundo digital; e **Internet**, esa conectividad que nos permite integrar el mundo físico en una red de comunicaciones, y convertirlo en servicios para poder interactuar con él desde cualquier lugar.

Android Things es la plataforma de Google para el desarrollo en dispositivos en entornos de Internet de las Cosas. Nos permite aplicar todo el potencial de Android a este nuevo mundo.

En este primer capítulo haremos una introducción general del sistema y realizaremos la instalación. Luego pasaremos a mostrar el hardware utilizado en el capítulo (Raspberry Pi). La parte más importante del capítulo es la descripción de los diferentes tipos de entrada/salida y cómo se programan. El capítulo acaba con la implementación de controladores de usuario y la integración de Google Assistant SDK en Android Things.

**Objetivos:**

- Mostrar en que consiste Android Things y cómo puede utilizarse para el desarrollo de aplicaciones de Internet de las cosas.
- Describir los mecanismos de seguridad y actualizaciones propuestos por Google.
- Enumerar las diferentes plataformas hardware soportadas, haciendo hincapié en Raspberry Pi.
- Describir el proceso de instalación y de las herramientas de desarrollo.
- Repasar algunos conceptos de electrónica usados en la unidad.
- Conocer las principales características y aprender a utilizar entradas / salidas. En concreto GPIO, PWM, I²C y UART.
- Comparar el uso de microcontroladores y microprocesadores a la hora de controlar sensores.
- Utilizar un microcontrolador programado con Arduino como esclavo de la Raspberry Pi.
- Aprender a implementar controladores de usuario.
- Integrar Google Assistant SDK en Android Things.

1.1. Internet de las cosas

Internet de las cosas (IoT), es una de las tendencias más prometedoras en tecnología. De acuerdo a muchos analistas, IoT puede ser la tecnología más activa en la próxima década. Tendrá un gran impacto en nuestras vidas y promete modificar nuestros hábitos.

La primera vez que se utilizó el término Internet de las Cosas fue en 1999 por Kevin Ashton, profesor del MIT. Utilizó la siguiente definición: "El IoT es el mundo en el que cada objeto tiene una identidad virtual propia y capacidad potencial para integrarse e interactuar de manera independiente en la red con cualquier otro individuo, ya sea una máquina o un humano."

Internet de las cosas incluye todos los objetos que pueden conectarse a Internet, para intercambiar información. Estos objetos pueden estar conectados en cualquier momento y en cualquier lugar. En resumen, IoT es un ecosistema donde los objetos están interconectados y, al mismo tiempo, se conectan a Internet.

El concepto de objetos conectados no es nuevo y con el paso de los años se ha desarrollado. El nivel de la miniaturización de circuitos y la potencia creciente de la CPU con un consumo menor hace posible que se imagine un futuro donde hay millones de "cosas" que se hablan entre sí.

Algunos ejemplos interesantes pueden ser el Amazon Dash Button, con solo pulsar un botón realizamos un pedido de un determinado producto (ver izq.). O el Amazon Key, que nos permite controlar la apertura de la puerta de casa a distancia (ver derecha).



Hay varios elementos que contribuyen a crear el ecosistema de IoT y es importante entender el papel que juegan. El componente básico de IoT es un objeto inteligente. Es un dispositivo que se conecta a internet y es capaz de intercambiar datos. Puede ser un sensor simple que simplemente toma un valor y lo transmite, o un sistema más complejo, con una mayor capacidad de procesamiento.

Todos los electrodomésticos de una casa son claros candidatos a convertirse en objetos inteligentes. Pero existen un abanico mucho más amplio wearables, automoción, vigilancia, ciudades inteligentes, ...

Existe una gran variedad de alternativas para que los objetos inteligentes obtengan conectividad: Ethernet, WiFi, Bluetooth, LoWPAN, NB-IoT, Zigbee, redes celulares, LoRA, ... También hay varios protocolos de aplicación ampliamente utilizados: HTTP, MQTT, CoAP, AMQP, XMPP, Stomp,...

1.2. Introducción a Android Things

Android Things es la nueva plataforma de Google para el desarrollo en dispositivos incrustados, que trabajan en entornos de Internet de las Cosas. Nos permite aplicar todo el potencial de Android a este nuevo mundo.

Android Things fue anunciado en Google I/O 2015 con nombre inicial de Brillo. Aunque en los primeros desarrollos tenía bastantes diferencias con Android (por ejemplo, la no posibilidad de usar Java), la versión finalmente lanzada en el Google I/O 2018, es prácticamente un sistema Android adaptado a plataformas basadas en microprocesador.

La característica más destacable sería la rapidez en poner en marcha un producto comercial, sin necesidad de tener conocimientos en el diseño de sistemas integrados. Esto es posible gracias a que nos ofrece una solución completamente administrada.

1.2.1. Solución completamente administrada

Desarrollar y comercializar un producto de IoT es un proceso muy complejo. Tenemos que dominar conceptos de electrónica, sistemas de microprocesadores,

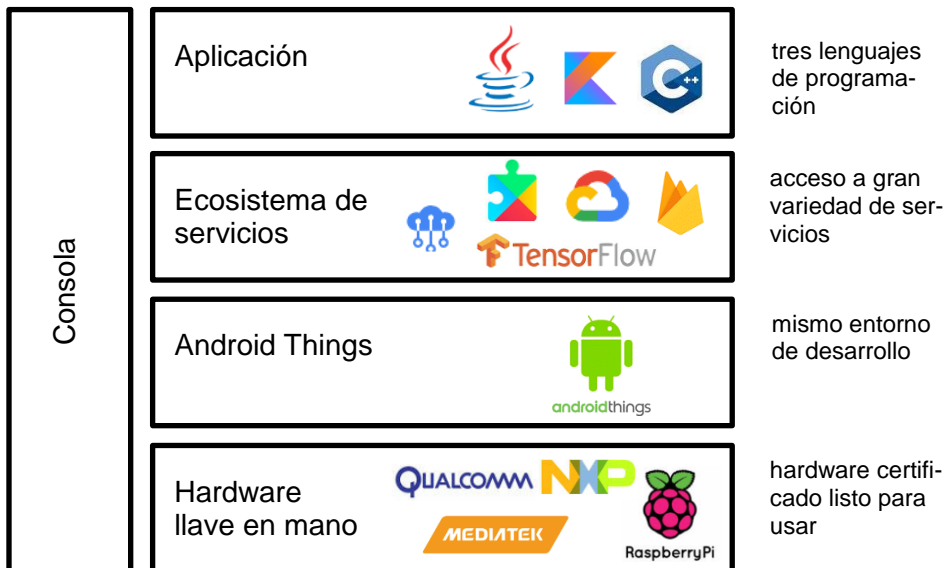
comunicaciones y software. Podemos destacar dos problemáticas particulares donde es especialmente complicado dar con una solución satisfactoria:

Actualizaciones de software: Va a ser inevitable que periódicamente tengamos que actualizar el software de nuestro producto para resolver errores en el desarrollo, agujeros de seguridad, mejoras de prestaciones, cambios de política de servicios o la realización de test A/B de usabilidad. Si dejamos las actualizaciones en manos de los usuarios, muchos nunca lo realizarán. Por lo que será más interesante realizar estas actualizaciones de forma transparente al usuario.

Seguridad: Un dispositivo de IoT puede recopilar información muy sensible para nuestros usuarios (cámaras, hábitos de vida, ...). Toda comunicación entre dispositivos o con la nube no ha de poder ser interceptada. Otra situación, si cabe más peligrosa es la posibilidad de que se introduzca software malicioso en el dispositivo. Tendríamos que garantizar que el software no ha sido reemplazado. Incluso en caso de que se tuviera acceso completo al dispositivo.

El ciclo de desarrollo de un producto que cumpla estas características suele ser elevado. Seleccionar el hardware, microprocesadores, protocolos, etc. podría demorarse meses o incluso años. Google nos ofrece una solución con la que podríamos disponer de un prototipo en función de semanas y llevarlo a fase de producción en cuestión de meses. Además, no resulta imprescindible tener conocimientos sobre sistemas integrados ni seguridad.

La solución está basada en una arquitectura de cuatro niveles:



En el nivel inferior tendríamos el hardware. Como veremos más adelante Google ha certificado cuatro plataformas (de NXP, Qualcomm y MediaTek) que podrán ser usadas en productos comerciales y dos solo para la fase de prototipo (de NXP y Raspberry Pi).

En el siguiente nivel encontramos el sistema operativo, por supuesto se trata de Android Things. Como se ha comentado no difiere demasiado de una distribución convencional. Simplemente se han añadido nuevas librerías para acceder a sensores y se han eliminado otras que no tendrían sentido en estos dispositivos.

En el siguiente nivel nos permite incorporar todos los servicios de Google o de terceros con los que estamos familiarizados en el desarrollo de aplicaciones Android. Entre estos servicios podemos destacar: Google Play Services, Google Cloud Platform, Cloud IoT Core, Firebase, TensorFlow, ...).

En el último nivel tenemos el nivel de aplicación, donde entra en juego nuestro desarrollo. Aunque existen algunas peculiaridades, como la gestión de permisos o el arranque de la aplicación, una aplicación para Android Things es básicamente igual que una para Android. Como se muestra en el gráfico disponemos de los mismos lenguajes de programación: Java, Kotlin o C++.

1.2.1.1. Modelo de actualizaciones

Para permitir que la solución sea completamente administrada juega un papel fundamental la consola. Una de las principales funciones de esta consola es subir las actualizaciones en línea (OTA) a los dispositivos. Estas actualizaciones pueden incluir la aplicación del fabricante o la imagen del sistema.

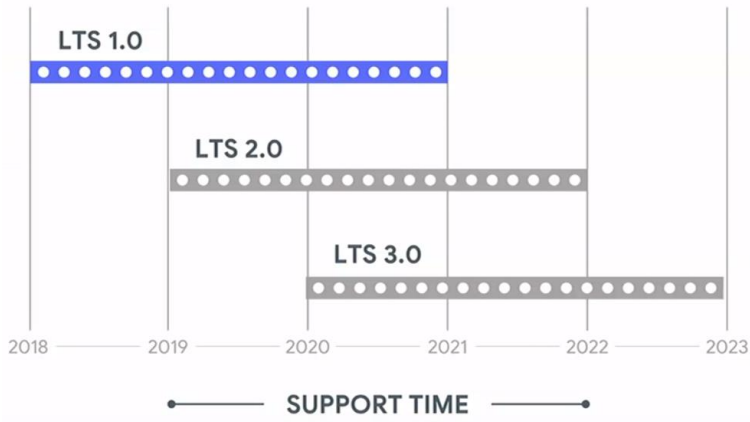
Google se compromete a hacerse cargo de las actualizaciones de forma gratuita durante los primeros tres años de soporte, siempre que ejecute la "versión de soporte a largo plazo" de Android Things. Las actualizaciones automáticas están habilitadas de manera predeterminada y llegarán como actualizaciones de seguridad mensuales y ocasionales actualizaciones importantes del sistema operativo. Google también menciona que después de tres años habrá "opciones adicionales para soporte extendido".

Si Google ha de hacer las actualizaciones le da un mayor control sobre el SO. En el modelo de actualizaciones tradicional de Android, las actualizaciones eran responsabilidad del fabricante. Esto era así porque este podía, y solía, modificar a su gusto el SO. A ningún fabricante le gusta un modelo de hardware interoperable (como el que se utiliza en los PC), donde el hardware es poco relevante. Todos quieren diferenciarse de la competencia y personalizar el software es un factor clave. El problema de este modelo es que muchos fabricantes no realizaban actualizaciones o dejaban de hacerlo en modelos más antiguos.

Google quiere cambiar esto, aprovechando su preponderancia en el mercado quiere forzar a los fabricantes a cambiar la política de actualizaciones. La idea es separar el sistema operativo Android de los controladores y firmware específicos del hardware en cada teléfono Android, esta iniciativa se conoce como "Proyecto Treble". Comenzando desde Android O, Google lanza actualizaciones de seguridad mensuales de Android directamente a los teléfonos, saltándose a los fabricantes.

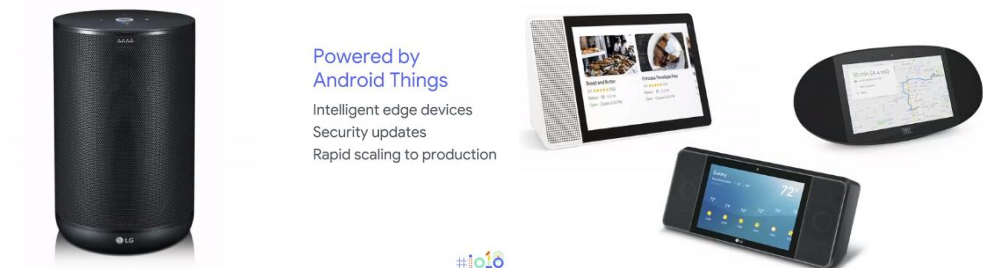
Las actualizaciones de seguridad están garantizadas por Google durante un periodo de tres años. De forma preestablecida, estas se producirán de forma automática, cada vez que Google lanza una actualización del sistema o nosotros actualizamos la aplicación. Las únicas actualizaciones que se realizarán de forma automática son las de la versión principal. Es decir, cuando salga la versión 2.0 de

Android Things, no estaremos obligados a utilizarla, pero seguiremos recibiendo actualizaciones para la versión 1.X.



1.2.1.2. Ejemplo de productos basados en Android Things

El primer producto comercial basado en Android Things es el altavoz inteligente LG WK7, que además de escuchar música permite interactuar con el asistente de Google. Otro tipo de dispositivo donde se está utilizando Android Things es en las pantallas inteligentes. Este concepto fue utilizado por primera vez en [Amazon Echo Show](#), básicamente añade una pantalla a un altavoz inteligente como el Amazon Echo o Google Home. Varias marcas, como LG, Lenovo y JBL, lanzarán en verano del 2018 sus modelos basados en Android Things. Como característica destacable de todos estos productos se puede destacar el breve periodo de tiempo desde que se concibe la idea hasta que el producto está en el mercado.



Vídeo[Tutorial]: *What's new in Android Things (Google I/O '18)*



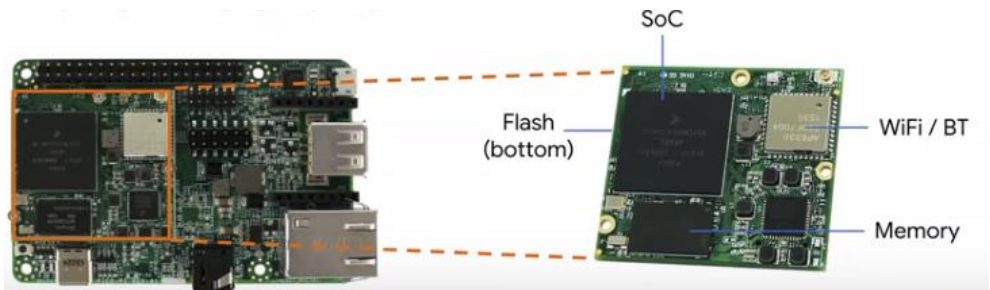
Preguntas de repaso: [Introducción a Android Things](#)

1.2.2. Plataformas hardware soportadas

Como hemos comentado Google trabaja con varios fabricantes para certificar que plataformas podrán recibir actualizaciones directamente desde Google.




Para definir estas plataformas se utiliza el concepto de SoM (System on Module) que es muy parecido al concepto de SoC (System on Chip). En ambos casos tenemos una CPU, RAM, almacenamiento, WiFi, E/S y todo lo demás que necesita para tener un ordenador, en un SoM se integra en una pequeña placa y en un SoC se integra en un chip de silicio. Los SoM son simplemente más grandes, y más baratos, que los SoC, ya que los componentes están esparcidos por una pequeña placa de circuito sin necesitar tanto nivel de integración.

Tanto si utilizamos un SoM como un SoC, es frecuente montarlos en una placa formando un SBC (Single Board Computer) donde se añaden E/S adicionales, conectores, alimentación, etc.



Ejemplo de un SoC, integrado en un SoM, integrado en un SBC.

En la siguiente tabla se muestra los cuatro SoM que actualmente soportan Android Things:

Platform	 Learn More	 Learn More	 Learn More	
CPU & Memory	<ul style="list-style-type: none"> • NXP i.MX8M • 1.5Ghz quad-core ARM Cortex A53 • 1GB or 2GB RAM 	<ul style="list-style-type: none"> • Qualcomm Snapdragon™ 212 • Quadcore 1.267Ghz ARM Cortex A7 • 1GB RAM 	<ul style="list-style-type: none"> • Qualcomm Snapdragon™ 624 • Octacore 1.8Ghz ARM Cortex A53 • 2GB RAM 	<ul style="list-style-type: none"> • MT 8516 • 1.3Ghz quad-core ARM Cortex A35 • 512MB RAM
GPU	QC7000Lite	QC Adreno 304	QC Adreno 506	N/A
Storage	4GB eMMC	4GB eMMC	4GB eMMC	4GB eMMC
Display	MX8-DSI-OLED1	N/A	8-inch WXGA Innolux Display with Touch	N/A
Camera	OV5640 MIPI CSI	N/A	Omnivision OV5693 5MP sensor	N/A
Networking	10/100/1000 Ethernet Wi-Fi 802.11ac Bluetooth® 4.2	Wi-Fi 802.11ac (2.4/5.0GHz) Bluetooth® 4.2	Wi-Fi 802.11ac (2.4/5.0GHz) Bluetooth® 4.2	10/100/1000 Ethernet Wi-Fi 802.11ac (2.4/5.0GHz) Bluetooth® 5.0
USB	2x USB 3.0 Type C	2x USB 2.0 Host 1x USB 2.0 OTG	1x USB 3.0 Type C	1x USB 2.0 Host 1x USB 2.0 OTG
Size (width x length)	50.3mm x 50.3mm	50mm x 46.5mm	50mm x 46.5mm	N/A
Type	Physical	Physical	Physical	Virtual

Adicionalmente se proporcionan plataformas de desarrollo para permitir la creación de prototipos y realizar las pruebas. No cumplen con los requisitos de seguridad de Google para la certificación de clave de identificación y arranque verificado, y no pueden recibir actualizaciones de seguridad:

Platform	NXP Pico i.MX7D	Raspberry Pi 3 Model B
CPU & Memory	<ul style="list-style-type: none"> NXP i.MX7D 1GHz dual-core ARM Cortex A7 512MB RAM 	<ul style="list-style-type: none"> Broadcom BCM2837 1.2GHz quad-core ARM Cortex A53 1GB RAM
GPU	N/A	VideoCore IV
Storage	4GB eMMC	MicroSD card slot
Display	LCD8000-43T VL050-80128NM-C01	HDMI Raspberry Pi Touch Display
Camera	OV5640 MIPI CSI	RPi Camera module v2
Networking	10/100/1000 Ethernet Wi-Fi 802.11ac (2.4/5.0GHz) Bluetooth® 4.1	10/100 Ethernet Wi-Fi 802.11n (2.4GHz) Bluetooth® 4.1
USB	1x USB 2.0 Host 1x USB 2.0 OTG	4x USB 2.0 Host
Size (width x length)	37mm x 40mm	85mm x 56mm (complete board)
Type	Physical	Physical

Los SoM pueden ser físicos o virtuales. En el primer caso un SoM es una placa que puede extraerse físicamente. En el segundo es un diseño de referencia proporcionado por el fabricante y certificado por Google. Luego podemos integrar los componentes individuales de ese diseño directamente en un producto. Los SoM virtuales son buenos para productos de gran volumen en los que se necesita flexibilidad y control para diseñar los componentes que se ajusten al diseño de la placa y las dimensiones del producto.

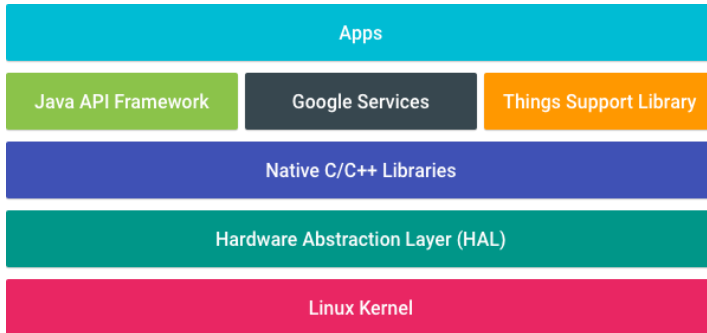
Para casa SoM Google nos proporciona un Board Support Package (BSP). Por lo que no es necesario interactuar con los fabricantes del hardware. Además, nos garantiza una sencilla migración en caso de cambiar de SoC.



Preguntas de repaso: [Plataformas hardware compatibles](#)

1.2.3. SDK de Android Things

El SDK de Android Things es muy similar al de Android. Básicamente es extendido con el módulo Things Support Library, que nos facilita el acceso al hardware.



Podemos destacar las siguientes diferencias con respecto a la programación para dispositivos móviles.

- Acceso más flexible a periféricos y controladores.
- Las aplicaciones del sistema no están presentes para optimizar los requisitos de inicio y almacenamiento.
- Los dispositivos solo muestran una aplicación a los usuarios, en lugar de múltiples como los dispositivos móviles. Esta aplicación es arrancada automáticamente al inicio. No se da al usuario la posibilidad de arrancar aplicaciones.

Dentro de Things Support Library se han añadido los siguientes APIs:

- **Bluetooth:** emparejamiento y conexión con dispositivos.
- **Device Updates:** actualizaciones de software basadas en OTA (Over The Air).
- **LoWPAN:** Acceso a redes de área personal inalámbricas de baja potencia ([LoWPAN](#))
- **NDK:** El Kit de Desarrollo Nativo se integra de forma predeterminada. Nos permite desarrollar las aplicaciones en C / C++.
- **Peripheral I/O:** Permite la comunicación con sensores y actuadores utilizando protocolos e interfaces estándar de la industria. Es compatible con GPIO, PWM, I2C, SPI y UART.
- **User Driver:** Los controladores de usuario nos permiten registrar nuevos drivers de dispositivo desde la aplicación. Podemos inyectar eventos de hardware al sistema, que otras aplicaciones podrán capturar.
- **Settings:** Nos permite configurar la pantalla, hora del sistema y configuraciones regionales.

Las siguientes características no serían soportadas en las APIs indicadas:

Característica	API
Interfaz de usuario (barra estado, navegación, quick settings)	NotificationManager KeyguardManager WallpaperManager
VoiceInteractionService	SpeechRecognizer
<code>android.hardware.fingerprint</code>	FingerprintManager
<code>android.hardware.nfc</code>	NfcManager
<code>android.hardware.telephony</code>	SmsManager TelephonyManager
<code>android.hardware.usb.accessory</code>	UsbAccessory
<code>android.hardware.wifi.aware</code>	WifiAwareManager
<code>android.software.app_widgets</code>	AppWidgetManager
<code>android.software.autofill</code>	AutofillManager
<code>android.software.backup</code>	BackupManager
<code>android.software.companion_device_setup</code>	CompanionDeviceManager
<code>android.software.picture_in_picture</code>	Activity Picture-in-picture
<code>android.software.print</code>	PrintManager
<code>android.software.sip</code>	SipManager

1.2.3.1. Cambios de comportamiento

Aplicaciones comunes no disponibles: No se incluyen muchas de las aplicaciones comunes en los teléfonos móviles, como el teléfono, contactos, Settings, calendario, ... Por lo tanto, no podremos usar las intenciones comunes que utilizaban estas aplicaciones o los proveedores de contenido estándar.

Salida por pantalla opcional: Algunos dispositivos disponen de salida HDMI donde podremos conectar una pantalla. Podremos diseñar la salida por pantalla de igual forma como lo hacíamos en Android. La ventana de la aplicación ocupa todo el espacio de la pantalla. Android Things no incluye la barra de estado del sistema ni los botones de navegación. Sin embargo, muchos de los sistemas a desarrollar no van a necesitar salida por pantalla. A pesar de este hecho, las actividades siguen siendo el componente principal de la aplicación. El sistema seguirá enviando los eventos de entrada a la actividad en primer plano, que tenga el foco.

Actividad de arranque (home): En Android Things no hay una aplicación de arranque (home), desde donde el usuario pueda lanzar otras aplicaciones. Se espera que una de las aplicaciones instaladas asuma este rol y se ejecute tras el arranque del sistema. Para ello, la aplicación ha de tener una actividad con un filtro de intención que incluya `CATEGORY_DEFAULT` y `IOT_LAUNCHER`.

Declaración de permisos: Los permisos necesarios han de ser declarados en el manifiesto. Los permisos peligrosos se otorgan al arrancar el dispositivo sin la verificación del usuario. Por lo tanto, tras instalar una aplicación con permisos peligrosos tendrás que reiniciar el sistema.

Notificaciones: No son soportadas. Al no existir la barra de estado, no podrían mostrarse.

Soporte para servicios de Google: Muchas de las Apis de Google tienen soporte para Android Things. Sin embargo, aquellas que requieren de una entrada directa del usuario o de su autenticación no van a ser soportadas. Cada versión de Android Things incluye la última versión estable de Google Play Services. Esta versión no podrá ser actualizada a través de Google Play Store, al no estar incluido en Android Things. En la siguiente tabla se muestran las Apis soportadas y no soportadas:

Servicios soportados	Servicios no soportados
Awareness, Cast, Google Analytics for Firebase, Firebase Authentication, Firebase Cloud Messaging (FCM), Firebase Crash Reporting, Firebase Realtime Database, Firebase Remote Config, Firebase Storage, Fit, Instance ID, Location, Maps, Nearby Connections, Nearby Messages, Places, Mobile Vision, SafetyNet	AdMob, Android Pay, Drive, Firebase App Indexing, Firebase Dynamic Links, Firebase Invites, Firebase Notifications, Play Games, Sign-In

1.2.4. Consola Android Things

Desde la consola de Android Things podremos instalar y actualizar la imagen del sistema en los dispositivos hardware. Nos permite:

- Descargar e instalar la última imagen del sistema Android Things.
- Crear imágenes de fábrica que contengan aplicaciones OEM junto con la imagen del sistema
- Instalar actualizaciones de la aplicación de forma online (OTA - On The Air).
- Monitorear las aplicaciones para comprobar qué están funcionando correctamente.

Para acceder a la consola: <https://partner.android.com/things/console/>

Más recursos: <https://androidthings.withgoogle.com>



Preguntas de repaso: [SDK de Android Things](#)

1.3. Raspberry Pi 3

En este capítulo vamos a trabajar con un dispositivo concreto, por lo que va a ser interesante conocer sus características y disponer de ciertos detalles que nos permitirán realizar las conexiones.

A este dispositivo se le conoce como *Low Cost Single Board Computers* o sistema embebido. Y suele emplearse para realizar una o algunas funciones dedicadas durante la fase de prototipo. Es decir, se trata de un ordenador de reducido precio, integrado en una pequeña placa que puede ser utilizado para resolver problemas concretos.

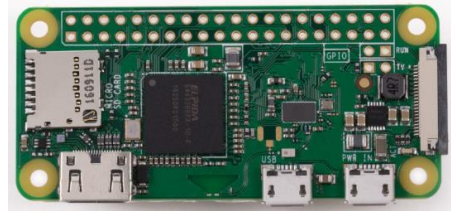
1.3.1. Comparativa con otros modelos

El modelo Raspberry Pi 3 es en la actualidad el más popular, pero existen gran variedad de modelos. Puedes consultar una tabla comparativa en¹. A continuación, destacamos algunos modelos:

Raspberry Pi 3 B (35€ PCcomponentes, 30€ Aliexpress)
4 x ARM Cortex-A53 a 1,2 GHz,
RAM 1G, Ethernet, Wifi, Bluetooth,
HDMI, 4xUSB 2.0, Jack audio,
ranura SD



Raspberry Pi Zero W (13,9€ Aliexpress)
Broadcom BCM2835 a 1 GHz,
RAM 512M, HDMI, 1x microUSB,
ranura SD



Orange Pi Plus 2 (12,3€ Aliexpress)
H3 Quad-core Cortex-A7 a 1,6GHz,
H.265/HEVC 4K, RAM 2G, Ethernet,
Wifi, Bluetooth, HDMI, 4xUSB
2.0, ranura SD, 16 G Flash



Intel Edison (91€)

2 x Atom Silvermont a 0,5GHz + 1 x
Quark a 0,1GHz, RAM 4G, Wifi,
Bluetooth, USB 2.0, ranura SD

<https://software.intel.com/es-es/iot/hardware/edison>



NOTA: Solo Raspberry PI 3 B e Intel Edison son compatibles con Android Things.

¹ <http://socialcompare.com/en/comparison/raspberry-pi-alternatives>

1.3.2. Características

La Raspberry Pi 3 está basada en el System on Chip (SoC) BCM2837 de Broadcom. Un SoC es un ordenador completo integrado en un chip, es decir, un procesador, RAM, GPU y demás funciones. Este modelo es un ARM de 64bits a 1.2GHz, quad-core y con 1GB de RAM. Se parece mucho a los SoC utilizados en los teléfonos inteligentes.



Este chip se monta en una placa de desarrollo junto a otros elementos: Un chip LAN9514 para controlar los puertos USB y Ethernet. Otra parte importante para este capítulo es que cuentan con una conexión GPIO (General Purpose Input Output), similares a las de los microcontroladores que solemos utilizar con Arduino. Eso sí, funcionan a 3.3v y no a 5v. El sistema operativo va en una tarjeta Micro SD. Si queremos que este sea Android Things tendrá que ser de al menos 8GB.



- 1.- cuatro puertos USB 2,0 hub
- 2.- Ethernet 10/100 Mbps (modelo+ 1000Mbps)
- 3.- Alimentación con conector micro USB (usar cargador de 2,5A)
- 4.- salida HDMI (full HD)
- 5.- conector de cámara CSI
- 6.- conector tipo Jack 3,5mm de salida de audio y vídeo compuesto
- 7.- ranura para tarjeta de memoria micro SD
- 8.- conector de pantalla DSI
- 9.- WiFi 802.11n y Bluetooth 4.1 (modelo+ 802.11ac dual band y Bluetooth 4.2 LS BLE)

10.- CPU Broadcom BCM2837 64bit quad-core a 1,2GHz, con 1 GB RAM (modelo+ BCM2837B0 a 1,4GHz)

11.- Controlador USB/Ethernet

12.- Conector de 40 pines con salidas GPIO

Una de las ventajas de este tipo de dispositivos es su gran versatilidad. Podemos adquirirlos para aprender a realizar nuestros proyectos y luego utilizarlos para otros propósitos, como un PC de escritorio, una consola de videojuegos, un centro multimedia para la TV o un router Wifi.

1.3.3. Alternativas para el Sistemas Operativos

Realmente Raspberry Pi es un ordenador de bajo coste y por lo tanto podremos instalar diferentes Sistemas Operativos (SO).

El SO más utilizado es [Raspbian](#). Podríamos decir que es el oficial, al haberse creado expresamente para este dispositivo. Se trata de una distribución de Linux basada en Debian. Hay otras distribuciones de Linux que también puedes instalar como [Fedora](#) o [Arch Linux](#), que disponen de versiones especiales para Raspberry Pi.

Aunque no hay una imagen oficial de Android para Raspberry Pi, existen alternativas de terceros para conseguirlo. Un ejemplo lo tenemos es [RaspAnd](#), con el que podremos ejecutar aplicaciones y juegos de Android. Incorpora el gestor de contenido Kodi y el navegador Firefox.

Una opción muy popular es instalar en la Raspberry Pi un gestor de contenido multimedia. Por ejemplo, [OSMC](#) o [OpenElec](#) (Open Embedded Linux Entertainment Center), ambos basado en Kodi. Si solo te interesa escuchar música instala [Pi MusicBox](#).

También se puede usar como NAS (Almacenamiento conectado en red (NAS) para disponer de nuestro sistema de almacenamiento en la nube. El chip que controla los puertos USB y la conexión Ethernet es el mismo (LAN9514), lo que produce un cuello de botella.

Otra opción muy popular es usarla como video consola. Con este fin podemos instalar [RetroPie](#), que nos permite ejecutar videojuegos antiguos de más de 50 sistemas. Además, permite instalar Kodi como reproductor multimedia.

También puedes instalar [Kano](#), un sencillo sistema operativo para que los niños tengan un primer contacto con la informática.

Para el desarrollo de dispositivos de Internet de las cosas, además de Linux, puedes instalar [Windows 10 IoT Core](#), la propuesta de Microsoft para este mundo. Necesitarás un PC con Windows y Visual Studio. Y por supuesto la alternativa que estudiaremos en este capítulo, [Android Things](#).



Preguntas de repaso: [Raspberry Pi 3](#)

1.4. Instalación de Android Things

Para realizar este apartado vas a necesitar el siguiente material:

- Raspberry Pi 3 B
- Tarjeta microSD con al menos 8GB
- Adaptador para leer tarjeta microSD en el PC
- Cable de alimentación microUSB
- Cable HDMI, un monitor y ratón (opcional)
- Cable Ethernet (opcional)

1.4.1. Descarga de la Imagen del Sistema de Android Things.



Ejercicio: Instalar Android Things en la memoria SD

Para poder disfrutar de Android Things en la Raspberry Pi vamos a tener que instalar el sistema operativo en la memoria SD.

NOTA: La realización de este ejercicio puede durar alrededor de una hora.

1. Descarga la herramienta de configuración en línea desde la consola:

<https://developer.android.com/things/preview/download.html>

En el menú selecciona la opción *Tools* y luego el botón *DOWNLOAD*.



Setup Utility

This command line tool will get you up and running with Android Things quickly. It will help you flash your board with either a generic image or your own custom image of Android Things and connect your board to Wi-Fi. A generic image is for early prototyping only and you will not have access to other Console features, such as metrics, crash reports, and over-the-air (OTA) updates. A custom image is for developers in the later stages of development beyond early prototyping.

DOWNLOAD

2. Descomprime el fichero y ejecuta la herramienta. En Windows, haz clic en el botón derecho en el archivo ejecutable y selecciona *Ejecutar como administrador*. En Mac o Linux, utiliza un comando similar al siguiente desde un terminal:

```
$ sudo ~ / Downloads / android - things - setup - utility / android -
things - setup - utility - linux
```

3. Selecciona la opción **1-Install Android Things and optionally set up Wi-Fi**

```
Android Things Setup Utility (version 1.0.19)
```

```
=====
```

```
This tool will help you install Android Things on your board and set up
Wi-Fi.
```

```
What do you want to do?
```

```
1 - Install Android Things and optionally set up Wi-Fi
```

2 - Set up Wi-Fi on an existing Android Things device

4. Selecciona la opción 1 (**Raspberry Pi 3**)

What hardware are you using?

- 1 - **Raspberry Pi 3**
- 2 - NXP Pico i.MX7D
- 3 - NXP Pico i.MX6UL

5. Selecciona de nuevo la opción 1:

Do you want to use the default image or a custom image?

- 1 - **Default image:** Used for development purposes. No access to the Android Things Console features such as metrics, crash reports, and OTA updates.
- 2 - Custom image: Upload your custom image for full device development and management with all Android Things Console features.

6. Tras descargar la imagen nos pedirá que introduzcamos la memoria SD en el ordenador y seleccionemos el adaptador:

Plug the SD card into your computer. Press [Enter] when ready

Running Etcher-cli...

? Select drive (Use arrow keys)

> \\.\PHYSICALDRIVE2 (8.0 GB) - SD/MMC Card Reader USB Device

7. Confirma que quieres borrar todos los datos de la SD. El proceso de instalación puede tardar hasta 20 minutos y el de verificación otro tanto.

? This will erase the selected drive. Are you sure? Yes

Flashing [=====] 100% eta 0s

Validating [=====] 24% eta 19m42s

NOTA: Si no usar esta herramienta de configuración, puedes descargar una imagen de Android Things (por ejemplo, desde la consola) y escríbala en la tarjeta microSD. Para ello puedes usar uno de estos links:

- [Windows](#)
- [Mac](#)
- [Linux](#)

8. Inserta la tarjeta microSD en la ranura en la parte inferior de la Raspberry Pi.

1.4.2. Configuración de la conexión a Internet.

Hasta que no lo conectemos a Internet no podremos decir que realmente es un dispositivo de IoT. La forma habitual es utilizar Ethernet o WiFi, pero también podríamos conectarnos por USB, Bluetooth, etc. Un dispositivo Android Things puede trabajar en condiciones muy diferentes. La forma más sencilla de conectarnos a Internet va a ser utilizando un monitor o un cable Ethernet. En ocasiones no tendremos esta posibilidad, a lo largo de este punto vamos ver diferentes alternativas para configurar la conexión.

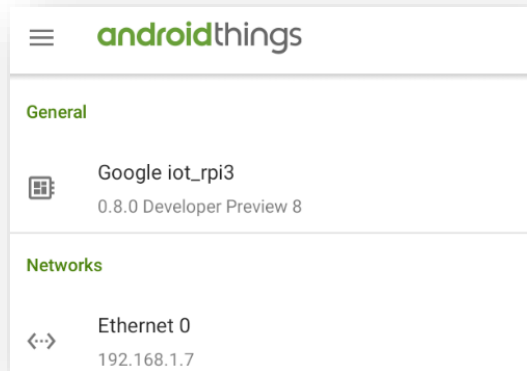


Ejercicio: Configurar WiFi con monitor

1. Conecta cables de alimentación al conector mini USB y el monitor al conector HDMI, tal como se muestra en la figura. El arranque puede durar unos minutos.



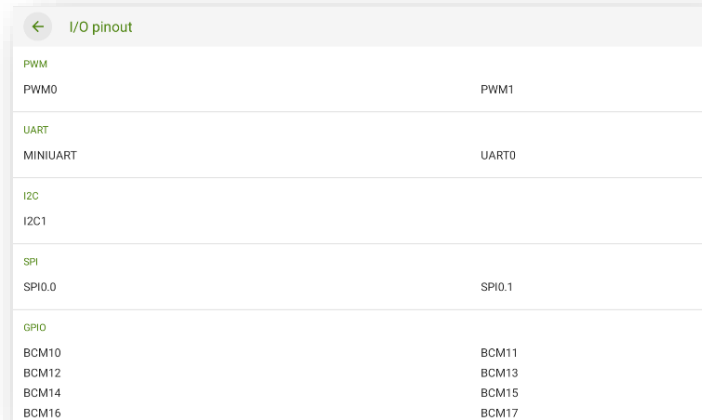
2. Si tienes un cable Ethernet conéctalo a tu red de área local. El servidor de DHCP asignará una IP al dispositivo. Podrás averiguar la IP que ha sido asignado a la Raspberry Pi, al ser mostrada en la pantalla:



3. Si no la has conectado por Ethernet puedes conectarla a una red WiFi. Conecta un ratón a un conector USB para poder interaccionar con la pantalla y selecciona la opción CONNECT TO NETWORK.

NOTA: Si no tienes acceso la red local por Ethernet, ni WiFi, puedes hacer una de las siguientes cosas: Conecta el cable Ethernet a tu ordenador y asigna a la Raspberry Pi una dirección IP usando DHCP. O conecta un cable serie de la Raspberry Pi a tu ordenador y usa una consola serie para conectarse a WiFi.

4. Desde la pantalla inicial dispones de tres opciones:
 - **General:** muestra la versión del sistema operativo, actualizar parches de seguridad, rearrancar el dispositivo o resetear con valores de fábrica.
 - **Networks:** permite configurar la conexión por Ethernet y WiFi.
 - **Peripherals:** muestra los periféricos, entre los que se incluyen los pines de entrada / salida (I/O Pinout):



I/O pinout	
PWM	
PWM0	PWM1
UART	
MINIUART	UART0
I2C	
I2C1	
SPI	
SPI0.0	SPI0.1
GPIO	
BCM10	BCM11
BCM12	BCM13
BCM14	BCM15
BCM16	BCM17



Ejercicio: Configurar WiFi usando Ethernet

En muchas ocasiones un dispositivo Android Things no está conectado a un monitor. En este caso vamos a disponer varias opciones para realizar la conexión a Internet. Veamos como conectarlo con un cable Ethernet.

1. Conecta la Raspberry Pi con un cable Ethernet conéctalo a tu red de área local. El servidor de DHCP asignará una IP al dispositivo. El problema es cómo averiguar la IP. Para ello puedes conectarte al switch y ver los dispositivos conectados con sus respectivas IP. Si no tienes acceso, puedes utilizar una herramienta de barrido de IPs, para ver las IPs asignadas en tu red (por ejemplo: <http://angryip.org/>).
2. Para asegurarte que es la IP correcta prueba el comando (cambiando la IP por la obtenida):

```
adb connect <dirección-ip>
```



Ejercicio: Configurar WiFi usando Setup Utility

A diferencia de Ethernet en WiFi vamos a necesitar configurar la red y contraseña desde la Raspberry Pi.

1. Ejecuta la herramienta Android Things Setup Utility usada en el ejercicio donde instalamos la imagen de las SD.
2. Selecciona la opción 2:

```
Android Things Setup Utility (version 1.0.19)
=====
```

This tool will help you install Android Things on your board and set up Wi-Fi.

What do you want to do?

- 1 - Install Android Things and optionally set up Wi-Fi
- 2 - Set up Wi-Fi on an existing Android Things device

3. Selecciona la opción 1:

What hardware are you using?

- 1 - Raspberry Pi 3
- 2 - NXP Pico i.MX7D
- 3 - NXP Pico i.MX6UL

4. Te pedirá que conectes el dispositivo mediante un cable Ethernet. Pulsa Enter y tratará de encontrarla de forma automática.
5. Si no la encuentra te pedirá que introduzcas su IP que ha obtenido a través de Ethernet.
6. Te pedirá el nombre SSID de la red y la contraseña para establecer la conexión.



Ejercicio: Configurar WiFi usando adb

Veamos un método alternativo para configurar WiFi en el dispositivo.

1. Abre una consola Shell en el dispositivo. Para ello tienes dos alternativas:
 - Asigna una IP usando un método alternativo. Ejecuta el comando `adb connect <dirección-ip>`.
 - Conecta un cable serie como se describe en el ejercicio: [Conexión por cable USB](#).
2. Ejecuta el comando `adb shell`. Este comando te permite abrir un shell Linux en el dispositivo.
3. Envía una intención para arrancar el servicio de Wi-Fi, indicando el SSID de tu red:

```
$ am startservice \
  -n com.google.wifisetup/.WifiSetupService \
  -a WifiSetupService.Connect \
  -e ssid <SSID_de_tu_red> \
```

Además, puedes añadir alguno de los siguientes parámetros:

- e `passphrase <contraseña>` Añádalo si tu red requiere contraseña.
- e `passphrase64 <contraseña_en_base64>` Añádalo si la contraseña contiene alguno de los siguientes caracteres `espacio, !, ", $, &, ', (,), ;, <, >, ` o |`. La contraseña ha de estar codificada en base64 (www.base64encode.org).
- ez `hidden true` Añádalo si el SSID de tu red es oculto.

NOTA: Introduce el comando en una línea, sin los caracteres `\`

4. Verifica que se ha conectado utilizando el `logcat`:

```
$ logcat -d | grep Wifi
```

```
...
V WifiWatcher: Network state changed to CONNECTED
V WifiWatcher: SSID changed: ...
I WifiConfigurator: Successfully connected to ...
```

5. Testea que tienes conectividad:

```
$ ping 8.8.8.8
```

6. Testea que la fecha y hora han sido configuradas a través de la red:

```
$ date
```

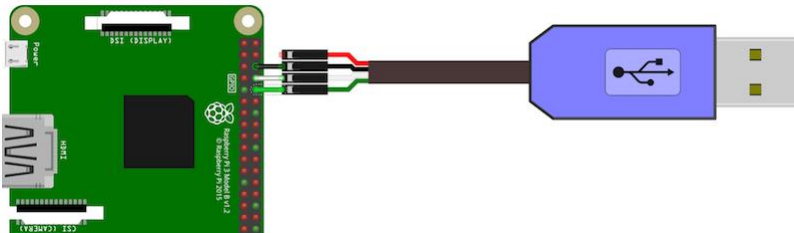
Una configuración incorrecta de fecha puede causar errores con SSL. Si es necesario reinicia el dispositivo.



Ejercicio: Conexión por cable USB

Si no dispones de conexión por WiFi o Ethernet existe una tercera posibilidad que consiste en usar un cable USB para conectarte a una consola serie. Podrás depurar el dispositivo y revisar la información de registro del sistema. Desde esta consola podrás ver los mensajes de registro del kernel (dmesg) y tendrás acceso a un shell completo que puede usar para acceder a comandos como `logcat`.

1. Consigue a un cable USB a TTL (<https://www.adafruit.com/products/954>):
2. Conéctalo como se muestra en la figura:



3. Conecta el USB a tu ordenador.
4. Abre un programa de terminal como PuTTY (Windows), Serial (Mac OS) o Minicom (Linux). Configura los parámetros del puerto serie siguientes: velocidad: 115200 baudios, bits de datos: 8, paridad: ninguna, bits de parada: 1.

1.4.3. Acceder al dispositivo desde Android Studio

Una vez conectado a Internet y conocida su IP podemos conectarlo con Android Studio para instalar y depurar aplicaciones.



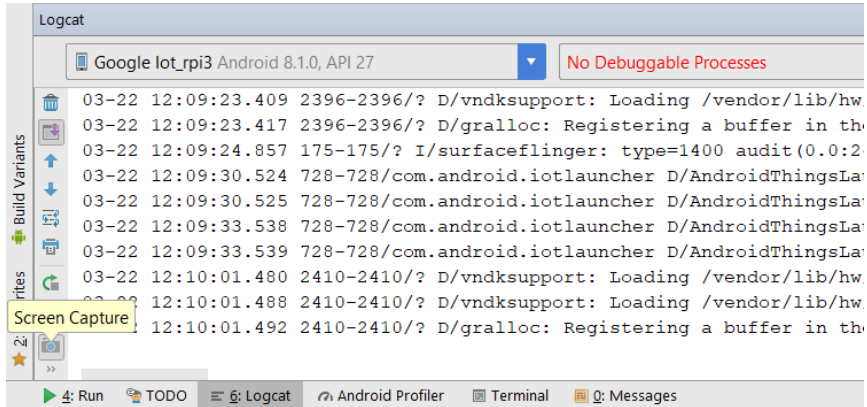
Ejercicio: Conectar el dispositivo con Android Studio

1. Desde una consola ejecuta el siguiente comando:

```
adb connect <dirección-ip>
```

Ha de aparecer `connect to <dirección-ip>`.

- Abre la pestaña Logcat y selecciona el dispositivo. Aparecerán todos los mensajes de Log:



- Selecciona el icono *Screen Capture* que se muestra en la imagen anterior. Se mostrará la pantalla del dispositivo. Esta opción es interesante cuando no dispongas de monitor.
- Abre la pestaña *Android Profiler* e investiga el estado de CPU, memoria y red.

1.4.4. Un primer proyecto.



Ejercicio: *Primer proyecto Android Things*

En este ejercicio aprenderás a crear tu primera aplicación para Android Things.

- Crea un nuevo proyecto con los siguientes datos:

Application name: *Android Things*

Package name: *org.example.androidthings*

☐ Phone and Tablet

☒ Android Things

Minimum SDK: API 24 Android 7.0 (Nougat)

Add an activity: *Android Things Empty Activity*

Activity Name: *MainActivity*

☒ Generate a UI layout File

Layout Name: *activity_main*

Deja el resto de opciones por defecto. Para tener acceso a las nuevas APIs es interesante que el target SDK sea API 27 Android 8.1 (Oreo) o superior.

- Navega por los ficheros generados y observa las diferencias.

- No se han añadido recursos para los iconos.
- En *AndroidManifest* dentro de la sección `<application>` se ha añadido la siguiente librería y un filtro especial para la actividad:

```
<uses-library android:name="com.google.android.things" />
<activity android:name=".HomeActivity">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.IOT_LAUNCHER" />
        <category android:name="android.intent.category.DEFAULT" />
    </intent-filter>
</activity>
```

- En *build.gradle (Module: app)* se ha añadido la siguiente dependencia:

```
dependencies {
    ...
    compileOnly 'com.google.android.things:androidthings:+'
}
```

3. Reemplaza el código de la actividad principal por:

```
public class HomeActivity extends Activity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        PeripheralManager perifericos = PeripheralManager.getInstance();
        Log.d("HomeActivity", "GPIO: " + perifericos.getGpioList());
    }
}
```

Este código obtiene una referencia del manejador de periféricos y muestra un log con la lista de las entradas/salidas GPIO disponibles.

4. Ejecuta el proyecto. El resultado ha de ser similar al siguiente:

```
com.example.androidthings D/HomeActivity: GPIO: [BCM10, BCM11, BCM12,
BCM13, BCM14, BCM15, BCM16, BCM17, BCM18, BCM19, BCM2, BCM20, BCM21,
BCM22, BCM23, BCM24, BCM25, BCM26, BCM27, BCM3, BCM4, BCM5, BCM6, BCM7,
BCM8, BCM9]
```



Práctica: Mostrar información en el Layout de la actividad.

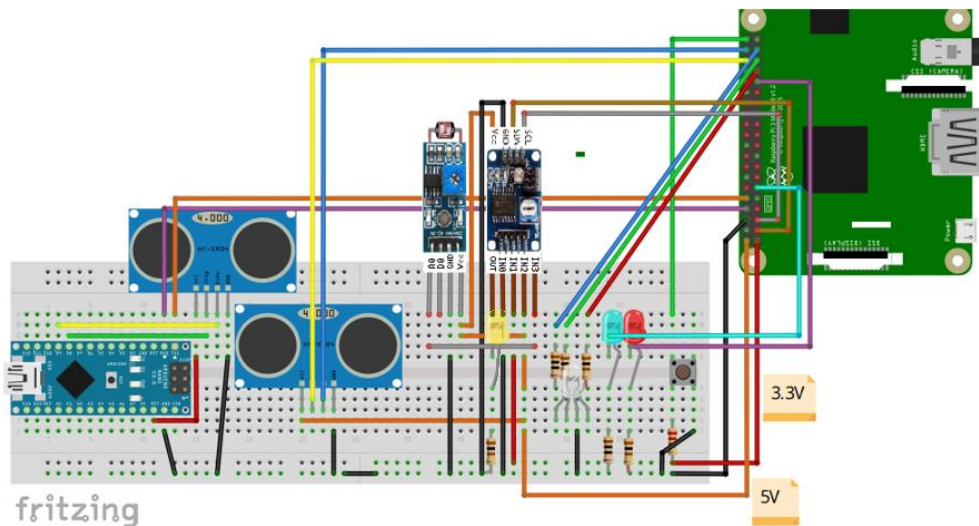
Utiliza el layout creado en la actividad para mostrar el resultado que hemos mostrado en el ejercicio anterior en el Logcat. Si conectaras un ratón, podrías incluso utilizar este Layout para realizar entradas en la aplicación.

1.4.5. Uso del laboratorio remoto

Lo ideal para la realización de estas unidades es que dispongas de una Raspberry Pi y los diferentes componentes para realizar los circuitos. No obstante, es posible que no tengas el material, o que te falte alguno de los componentes necesarios para realizar algún ejercicio concreto. En estos casos podrás utilizar el laboratorio remoto. Solo necesitas Android Studio, un navegador y una conexión a internet.

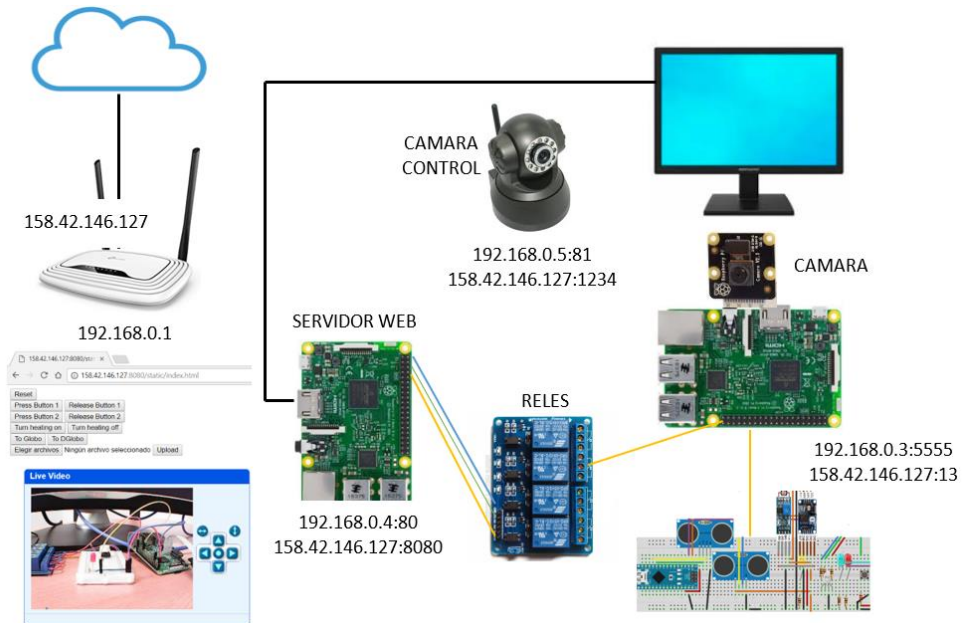
El primer paso ha de ser verificar si el laboratorio ya está siendo usado por otro alumno. Entra en Poliformat y selecciona la asignatura. En el menú de la izquierda selecciona la herramienta de Chat. Si nadie lo está usando escribe el mensaje “COMIENZO A USAR LAB. REMOTO”. Es muy importante que cuando termines el trabajo escribas “DEJO DE USAR LAB. REMOTO”. El periodo de uso es de una hora. Pasado este tiempo verifica en el Chat si algún compañero ha escrito un mensaje indicando que quiere usar el laboratorio. En caso de no ser así, escribe “CONTINUO USANDO LAB. REMOTO” y podrás utilizarlo una hora más.

El siguiente esquema muestra los circuitos montados en el laboratorio. Corresponde a los puntos 1.6 y 1.7.



NOTA: si no vas a usar el laboratorio remoto porque dispones de tu placa, te recomendamos que utilices una distribución parecida a la anterior. De esta forma, en lugar de montar y desmontar cada circuito, podrás montarlos todos en la misma placa.

El esquema de los diferentes elementos del laboratorio se muestra a continuación:



Los elementos son:

Servidor Web: Permite el acceso remoto para mostrar la imagen de la cámara y dispone de varios botones para realizar distintas acciones. Controla 4 relés que te permitirán resetear la Raspberry Pi y accionar botones de entrada.

Raspberry Pi con Android Things: Es el dispositivo que vas a programar.

Cámara de control: Nos ofrece una vista en tiempo real del laboratorio y permite verificar que la activación de los LEDs es correcta.

Los diferentes elementos se encuentran en una red que utiliza direcciones IP privadas en la red 192.168.0.0/24. Para acceder desde el exterior se utiliza la dirección IP pública 158.42.146.127. Cada elemento se mapea en un puerto distinto.

Para utilizar el laboratorio sigue los siguientes pasos:

- Con un navegador Web accede a <http://158.42.146.127:8080>.
- Verifica que tienes visión de los diferentes dispositivos y que están conectados. Puedes utilizar las flechas para cambiar el ángulo de la cámara.
- Si pulsas el botón de RESET la Raspberry Pi se reiniciará (no operativo).
- El botón UNINSTALL ALL desinstala todas las apps. Utilízalo si tienes problemas para instalar tus apps.
- Dispones de dos botones conectados a dos entradas GPIO de la Raspberry Pi.
- En un futuro se podrá poner en marcha un ventilador o una luz.

- Para realizar aplicaciones de visión artificial la Raspberry Pi dispone de una cámara que apunta a un monitor. Puedes cambiar la imagen mostrada en el monitor por medio de los botones <120>, <100>, <80>...
- Para interactuar con el dispositivo, desde la línea de comando escribe:

```
adb connect 158.42.146.127:13
```

- Abre Android Studio y verifica que en la pestaña Logcat aparece el dispositivo Google IoT_RPi3.
- En esta ventana selecciona el icono Screen Capture para verificar la salida gráfica del dispositivo.

NOTA: en la mayoría de prácticas esta salida no es relevante.

- Si seleccionas la pestaña Android Profiler podrás realizar un análisis de CPU, memoria y red usadas por el dispositivo.
- También es posible utilizar la pestaña Debug para realizar una depuración de tus aplicaciones.
- Es el momento de probar las diferentes prácticas. Para verificar su correcto funcionamiento utiliza la cámara que se muestra en la página Web u observa las salidas mostradas en el Logcat.

1.5. Algunos conceptos de electrónica

A lo largo de este capítulo vamos a tener que montar pequeños circuitos electrónicos para conectar sensores y actuadores. Es posible que tus conocimientos de electrónica sean limitados, si es así, esta sección puede ayudarte a comprender algunos aspectos claves.

Puedes leer esta sección ahora, pero también puedes esperar a que los diferentes conceptos sean necesarios en algún ejercicio. En estos casos se te indicará el apartado a leer.

1.5.1. Voltaje y fuente de alimentación

Los circuitos electrónicos manipulan la información representándola por medio de señales eléctricas. Normalmente es el voltaje o tensión eléctrica la magnitud utilizada para este propósito.

Para conseguir este voltaje vamos a necesitar una fuente de alimentación, por ejemplo, la Raspberry Pi utiliza un cargador USB. Veamos algunos voltajes utilizados en casi todos los circuitos:

V_{IN}: Voltaje de la fuente de alimentación conectado a la placa. En la Raspberry Pi 9V.

V_{CC} (o **V_{DD}**): Voltaje de corriente continua regulado interno que alimenta los componentes de la placa. Los voltajes comunes son + 5V, + 3.3V y + 1.8V.

GND: (Ground): Punto de referencia utilizado para indicar los 0V de la placa. Todos los voltajes se miden respecto a este. Se conoce también como masa.

1.5.2. Señales analógicas y digitales

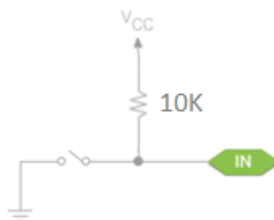
Según como puede variar el voltaje de la señal podemos diferenciar dos tipos de señales.

Analógicas: El voltaje de la señal es proporcional a la información que representa. Por ejemplo, si un sensor de temperatura nos da un voltaje entre 0V y 5V, que representa una temperatura entre 0° y 100°.

Digitales: El voltaje que puede tomar la señal se limita a un número finito de valores. Si estos valores son 2, se conocen como señales binarias. Cuando el valor está cercano a V_{CC} , se conoce como "1". Cuando el valor está cercano a GND, se conoce como "0". Para representar una magnitud como la temperatura se necesitan varias señales binarias, por ejemplo 8 bits, 00101100.

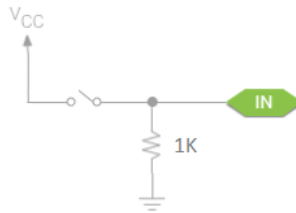
1.5.3. Resistencia pull-up / pull-down.

En el ejercicio anterior hemos conectado una resistencia entre la entrada GPIO y V_{CC} , tal y como se muestra en el esquema siguiente. Veamos por qué esta resistencia es necesaria. Si eliminaras esta resistencia y el pulsador estuviera abierto, tendríamos un cable desde la entrada, sin conectar a ningún sitio. Realmente no estaría conectado a masa, ni a V_{CC} , por lo que no podríamos decir que es ni 0 ni 1. A este tercer estado se le conoce como alta impedancia.



Puede ser peligroso dejar una entrada en alta impedancia. Un cable sin conectar, puede actuar como una antena. Si hay señales radio eléctricas cerca (producidas por un motor, antena...) se podría inducir una corriente eléctrica en este cable, que podría engañar al GPIO provocando su activación. Para evitar dejar la entrada en alta impedancia, la conectamos a V_{CC} . Pero, cuando se cierre el pulsador la corriente pasará de masa a V_{CC} directamente, provocando un cortocircuito. Al colocar esta resistencia, la corriente entra en la entrada, siguiendo el camino más fácil.

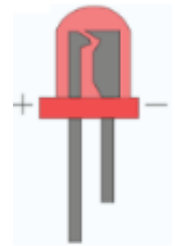
Si queremos que cuando el pulsador esté abierto el valor de la entrada sea 0, y al pulsarlo pase a 1, utilizaríamos una resistencia de pull-down.



El valor habitual para una resistencia de pull-up es de 10K y de pull-down es de 1K.

1.5.4. LED y cálculo de resistencia de ajuste

Se trata de un diodo emisor de luz (Light Emitting Diode). Como todo diodo la corriente solo puede pasar del ánodo (pata más larga) al cátodo (pata más corta). Por lo tanto, para que emita luz es imprescindible que conectes el ánodo a tensión alta y el cátodo a masa. **IMPORTANTE:** nunca conectes un LED directamente con un pin a 3,3/5V y el otro a GND. La corriente sería demasiado alta y el LED se fundiría. Has de limitar el paso de corriente utilizando una resistencia.



Para calcular el valor de la resistencia podemos utilizar la ley de Ohm: $V = I \cdot R$. El valor máximo de corriente que puede pasar por un LED es de 20mA (un valor de 17mA es suficiente, si es menor brillará algo menos). También has de tener en cuenta que un led provoca una caída de voltaje entre 1,8V - 2,1V si es rojo, amarillo o verde y entre 3V - 3,8V si es azul, violeta o blanco.

Un ejemplo: tenemos un LED rojo, con una caída de 1,8V y queremos que pase una corriente de 17mA. Si la fuente de alimentación es de 9V la resistencia necesaria sería:

$$V = I \cdot R \Rightarrow R = V / I = (9V - 1,8V) / 17mA = 423,5\Omega \approx 470 \Omega$$

Para una alimentación de 3,3V con LED rojo, con caída de 1,8V:

$$R = (3,3V - 1,8V) / 17mA = 88,2\Omega \approx 100\Omega$$

Para una alimentación de 3,3V con LED azul, con caída de 3,1V::

$$R = (3,3V - 3,1V) / 17mA = 11,8\Omega \approx 10\Omega$$

Más información sobre conceptos básicos de electrónica en: <https://developer.android.com/things/hardware/hardware-101.html>



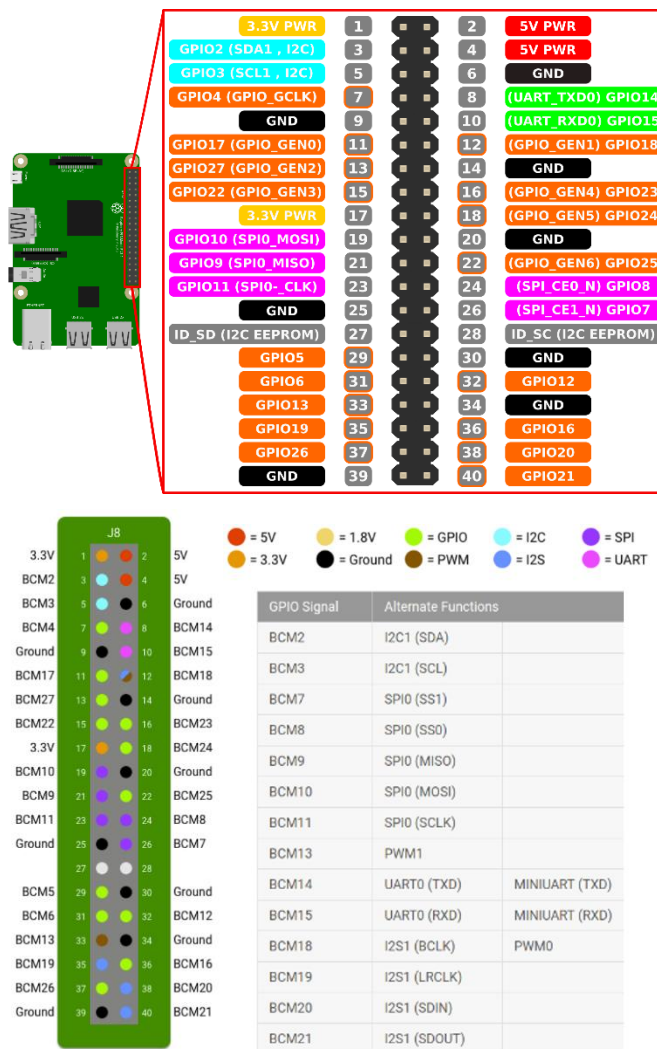
Preguntas de repaso: [Conceptos de electrónica](#)

1.6. Entradas / Salidas en Android Things

El potencial de Android Things está en la posibilidad de conectar hardware de entrada salida y poderlo controlar directamente desde nuestra la aplicación.

Las entradas salidas de propósito general se controlan por una serie de registros del chip BCM2837. Estos registros son mapeados en direcciones de memoria para su acceso. Cuando trabajas con un sistema operativo el acceso a estas posiciones está prohibido (excepto root) y tienes que utilizar los servicios ofrecidos por el sistema operativo.

Las conexiones a las entradas/salidas se realizan a través de 40 pines, tal y como se muestra a continuación:



Una opción interesante puede ser utilizar un cable que permite llevar estos 40 pines a la placa de prototipos. De esta forma protegemos el desgaste de la placa original.

GND Ground (8 terminales) – Toma de tierra. SE utiliza como referencia para el resto de voltajes.

5V (2 terminales) – Conectados a la alimentación. Se utiliza un fusible para evitar problemas en caso de cortocircuito. Podemos consumir hasta 1A desde estos pines.

3,3V (2 terminales) – Podemos consumir hasta 1A.

GPIO (24 terminales) – Cada pin puede configurarse para trabajar como entrada o como salida. Como entrada hay que usar una tensión entre 0 y 3,3V. Las salidas no tienen que tener un consumo superior a 50mA.

PWM (2 terminales)

UART (2 terminales, 1 bus)

I²C (4 terminales, 2 buses)

SPI (5 terminales, 2 buses)

1.6.1. Conexiones GPIO

GPIO es el acrónimo de General Purpose Input/Output, en castellano Entrada/Salida de Propósito General. Corresponde a un pin de un chip que puede utilizarse para realizar una entrada o salida binaria, normalmente para interactuar con algún dispositivo exterior.

Si lo programamos como salida e indicamos nivel bajo en este pin tendremos 0V, y si indicamos nivel alto tendremos un voltaje de 3,3V. Hay que tener la precaución de no conectar a esta salida un dispositivo con consumo superior a 16mA. El máximo que puede proporcionar todas las salidas simultáneamente es 50mA.

Si por el contrario lo programamos como entrada, leeremos un 0 si la tensión es menos de 0,8V y leeremos un 1 si la tensión está entre 1,3 y 3,3V. Nunca hay que superar este voltaje, dado que podríamos destruir el bloque GPIO.

Podemos tener hasta 24 entradas/salidas GPIO, pero algunos de los pines pueden compartir otras funciones como I2C, UART o PWM.

Hay que destacar que en Raspberry Pi no hay entradas analógicas como en otros controladores (como muchos compatibles Arduino). Si queremos leer un valor analógico tendremos que utilizar un conversor Analógico/Digital (A/D), como veremos más adelante.

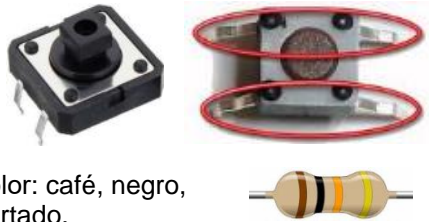


Ejercicio: *Entrada GPIO en Android Things*

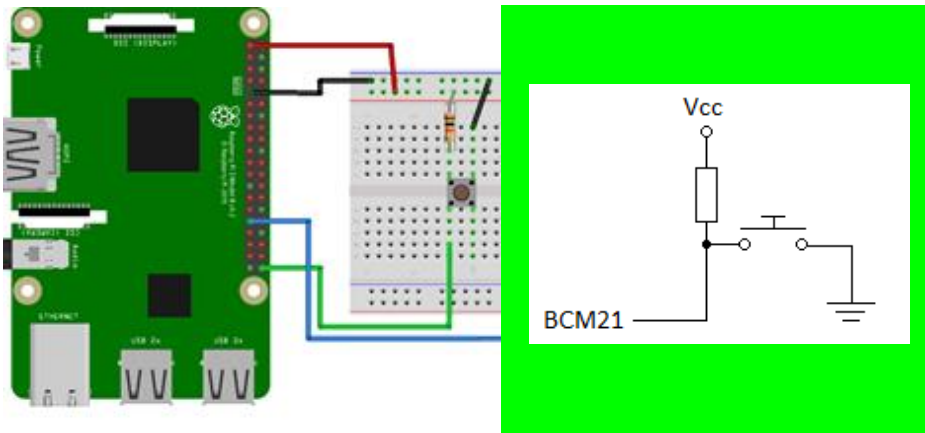
En este ejercicio aprenderás a comunicarte con el hardware del dispositivo. En concreto, leer la entrada de un simple botón.

Material necesario:

- un pulsador:
Si utilizas un pulsador de 4 pines, ten en cuenta que los pines marcados están interconectados. Al pulsar, los 4 pines estarían unidos.
- una resistencia de pull-up de 10 K Ω (color: café, negro, naranja). Se explica en el siguiente apartado.
- tablero de prototipos y cables.



1. Monta los sensores y actuadores sobre la placa de prototipos y conectarlos a la Raspberry Pi. El siguiente esquema te muestra cómo hacerlo:



- Conecta una conexión del botón al pin de entrada GPIO elegido (en este ejemplo BCM21).
- Conecta el otro extremo del botón a tierra y el mismo pin de entrada GPIO a **+3.3V** a través de una resistencia pull-up de 10 K Ω .

2. Añade el siguiente código a `HomeActivity` para controlar la pulsación del botón:

```
private static final String BOTON_PIN = "BCM21"; // Puerto GPIO del botón
private Gpio botonGpio;

@Override protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    PeripheralManager manager = PeripheralManager.getInstance();
    try {
        botonGpio = manager.openGpio(BOTON_PIN); // 1. Crea conexión GPIO
        botonGpio.setDirection(Gpio.DIRECTION_IN); // 2. Es entrada
        botonGpio.setEdgeTriggerType(Gpio.EDGE_BOTH);
        // 3. Habilita eventos de disparo por flanco de bajada
        botonGpio.registerGpioCallback(callback); // 4. Registra callback
    } catch (IOException e) {
        Log.e(TAG, "Error en PeripheralIO API", e);
    }
}
```

```

    }

    private GpioCallback callback = new GpioCallback() {
        @Override public boolean onGpioEdge(Gpio gpio) {
            try {
                Log.e(TAG, "cambio botón "+Boolean.toString(gpio.getValue()));
            } catch (IOException e) {
                e.printStackTrace();
            }
            return true; // 5. devolvemos true para mantener callback activo
        }
    };

    @Override protected void onDestroy() {
        super.onDestroy();
        if (botonGpio != null) { // 6. Cerramos recursos
            botonGpio.unregisterGpioCallback(callback);
            try {
                botonGpio.close();
            } catch (IOException e) {
                Log.e(TAG, "Error al cerrar botonGpio.", e);
            }
        }
    }
}

```

Para conseguir reaccionar ante el cambio de una entrada GPIO hemos seguido los siguientes pasos:

1. Usamos `PeripheralManager` para abrir una conexión con el puerto GPIO conectado al botón.
2. Configuramos el puerto como de entrada.
3. Configuramos qué transiciones de estado generarán eventos de llamada. Puede ser `EDGE_NONE` (no se dispara), `EDGE_RISING` (por flanco de subida), `EDGE_FALLING` (por flanco de bajada), `EDGE_BOTH` (por flanco de subida y bajada).
4. Registra un objeto `GpioCallback` para recibir los eventos de disparo.
5. Implementamos el objeto callback. El método `onGpioEdge(Gpio)` será llamado cuando ocurra el evento programado. El parámetro que se pasa es el puerto GPIO que causa el evento. Resulta sencillo averiguar su nombre de puerto (`getName()`) o su valor (`getValue()`). Importante, este método ha de devolver verdadero, si queremos continuar recibiendo futuros eventos de disparo.
6. Cuando la aplicación ya no necesita la conexión GPIO, cierra el recurso. En el ejemplo lo hacemos en `onDestroy()`.

3. Solicita el siguiente permiso en AndroidManifest:

```

<uses-permission
    android:name="com.google.android.things.permission.USE_PERIPHERAL_IO"/>

```

4. Ejecuta el proyecto. Se producirá un error de compilación. Si abres el Logcat verás que se indica que el permiso no ha sido otorgado. Reinicia la Raspberry Pi para que el permiso sea otorgado.

5. Ejecuta de nuevo el proyecto. Cada vez que sueltes el botón ha de aparecer una línea en Logcat:

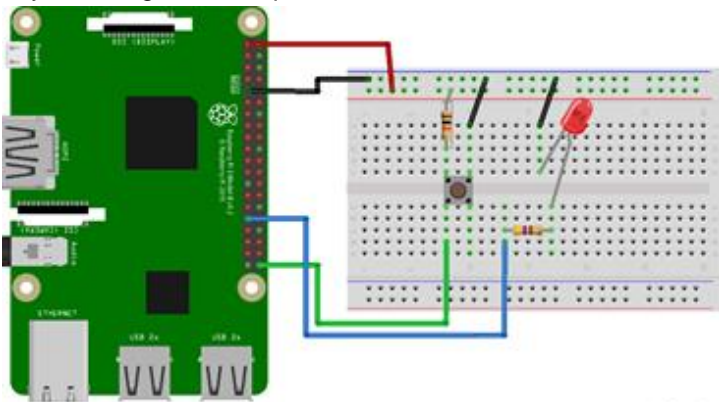


Ejercicio: Salida GPIO en Android Things

En este ejercicio aprenderás a comunicarte con una salida GPIO. En concreto un simple LED. Programaremos un **Handler** que se ejecute cada segundo para conseguir un efecto de parpadeo en el LED.

Material adicional:

- **un LED:** de color rojo.
 - Resistencia de 100 Ω (color: marrón, negro, marrón)
1. Monta los sensores y actuadores sobre la placa de prototipos y conectarlos a la Raspberry Pi. El siguiente esquema te muestra cómo hacerlo:



fritzing

- Conecta el pin de salida GPIO elegido (BCM6) a un extremo de una resistencia en serie.
- Conecta el otro extremo de la resistencia al ánodo del LED (pata más larga).
- Conecta el cátodo del LED (pata más corta) a tierra.

2. Añade el siguiente código a **HomeActivity** para controlar el LED:

```
private static final int INTERVALO_LED = 1000; // Intervalo parpadeo (ms)
private static final String LED_PIN = "BCM6"; // Puerto GPIO del LED
private Handler handler = new Handler(); // Handler para el parpadeo
private Gpio ledGpio;

@Override protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    PeripheralManager manager = PeripheralManager.getInstance();
    ...
    try {
        ledGpio = manager.openGpio(LED_PIN); // 1. Crea conexión GPIO
        ledGpio.setDirection(Gpio.DIRECTION_OUT_INITIALLY_LOW);
        // 2. Se indica que es de salida
    }
}
```

```

        handler.post(runnable); // 3. Llamamos al handler
    } catch (IOException e) {
        Log.e(TAG, "Error en PeripheralIO API", e);
    }
}

private Runnable runnable = new Runnable() {
    @Override public void run() {
        try {
            ledGpio.setValue(!ledGpio.getValue()); // 4. Cambiamos valor LED
            handler.postDelayed(runnable, INTERVALO_LED);
            // 5. Programamos siguiente llamada dentro de INTERVALO_LED ms
        } catch (IOException e) {
            Log.e(TAG, "Error en PeripheralIO API", e);
        }
    }
};

```

Para conseguir reaccionar ante el cambio de una entrada GPIO hemos seguido los siguientes pasos:

1. Usamos `PeripheralManager` para abrir una conexión con el puerto GPIO conectado al LED.
2. Configuramos el puerto como de salida, indicando que inicialmente esté a nivel bajo.
3. Usando el `Handler`, hacemos una primera llamada al objeto `Runnable`. El código de este objeto se ejecutará en un nuevo hilo.
4. El valor de salida es cambiado. Si estaba activo lo desactivamos y a la inversa.
5. Programamos una nueva llamada al `Runnable` para dentro de un segundo. Con lo que conseguiremos un efecto de parpadeo.

3. Ejecuta el proyecto. Verifica que el LED parpadea.



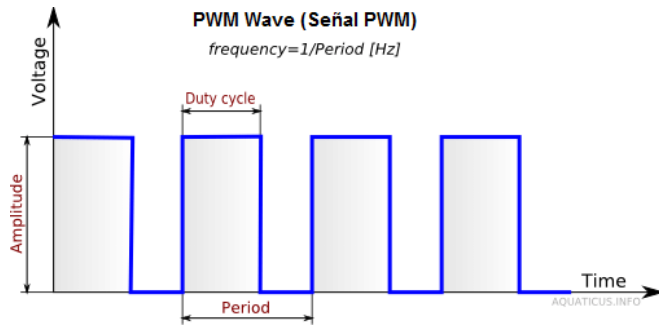
Práctica: Controlar parpadeo mediante el botón.

Tras realizar los dos ejercicios anteriores, modifica el proyecto para que el LED deje de parpadear cuando el botón está pulsado.

1.6.2. Salidas PWM

Como acabamos de ver, una salida GPIO solo puede tomar dos valores, 1 o 0. Imaginemos que queremos iluminar un LED, pero con una salida algo menor que 1 para que brille menos. Una solución podría ser utilizar una salida analógica para controlar el voltaje exacto que queremos aplicar. Dado que Raspberry Pi no dispone de salidas analógicas, tendríamos que utilizar un conversor D/A (digital analógico).

Existe otra solución que consiste en intercalar periodos de 1 con periodos de 0 en la salida. Si lo hacemos con una frecuencia lo suficientemente alta el efecto no será perceptible y obtendremos el resultado deseado.



La modulación de ancho de pulso o PWM² (Pulse Width Modulation) es un método común usado para aplicar una señal de control proporcional a un dispositivo usando una salida binaria. Por ejemplo, los servomotores usan PWM para determinar su velocidad de rotación. O las pantallas LCD ajustan su brillo en función del valor promedio de una señal PWM.

Una señal PWM es controlado por medio de dos parámetros. El periodo de la señal y el porcentaje del tiempo en el que la señal está a nivel alto.

La Raspberry Pi solo dispone de dos salidas PWM: Sus nombres son “PWM0” (pin 12, segunda función de “BCM18”) y “PWM1” (pin 33, segunda función de “BCM13”).



Ejercicio: Salida PWM en Android Things

En este ejercicio aprenderás a configurar una salida PWM. En concreto vamos a configurar el nivel de brillo de un LED.

Material específico necesario:

- un LED azul
- resistencia de 10 Ω (color: marrón, negro, negro)



1. Conecta el ánodo de un LED azul a la salida P18 (BCM18), conecta el cátodo a masa por medio de una resistencia de 10 Ω .

2. Añade el siguiente código a `HomeActivity` para controlar el LED por PWM:

```
private static final int PORCENTAGE_LED_PWM = 25; // % encendido
private static final String LED_PWM_PIN = "PWM0"; // Puerto del LED
private Pwm ledPwm;

@Override protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    PackageManager manager = PackageManager.getInstance();
    ...
    try {
```

² <https://developer.android.com/things/sdk/pio/pwm.html>

```

        ledPwm = manager.openPwm(LED_PWM_PIN); // 1. Crea conexión GPIO
        ledPwm.setPwmFrequencyHz(120);        // 2. Configuramos PWM
        ledPwm.setPwmDutyCycle(PORCENTAGE_LED_PWM);
        ledPwm.setEnabled(true);
    } catch (IOException e) {
        Log.e(TAG, "Error en al acceder a salida PWM", e);
    }
}

@Override protected void onDestroy() {
    super.onDestroy();
    ...
    if (ledPwm != null) {                        // 3. Cerramos recursos
        try {
            ledPwm.close();
            ledPwm = null;
        } catch (IOException e) {
            Log.e(TAG, "Error al cerrar PWM", e);
        }
    }
}
}

```

Para conseguir reaccionar ante el cambio de una entrada GPIO hemos seguido los siguientes pasos:

1. Usamos `PeripheralManager` para abrir una conexión con el puerto PWM conectado al LED.
2. Configuramos la frecuencia de la señal generada y el porcentaje en que estará activa la señal.
3. Cuando la aplicación ya no necesita la salida PWM, cierra el recurso. En el ejemplo lo hacemos en `onDestroy()`.
3. Ejecuta el proyecto. Verifica la intensidad LED. Aumenta el valor de `PORCENTAGE_LED_PWM` para aumentar su brillo.
4. Introduce un valor de `DutyCycle` de 25 y una frecuencia de 1. El LED se encenderá cada segundo, durante 250 mseg.



Práctica: **Cambiar el brillo de un LED de forma periódica**

Aprovechando el temporizador de un segundo utilizado para el LED rojo, haz que el led azul modifique su brillo siguiendo la siguiente secuencia: 0%. 20%, 40%, 60%, 80%, 100%, 0%. 20%, 40%, ..

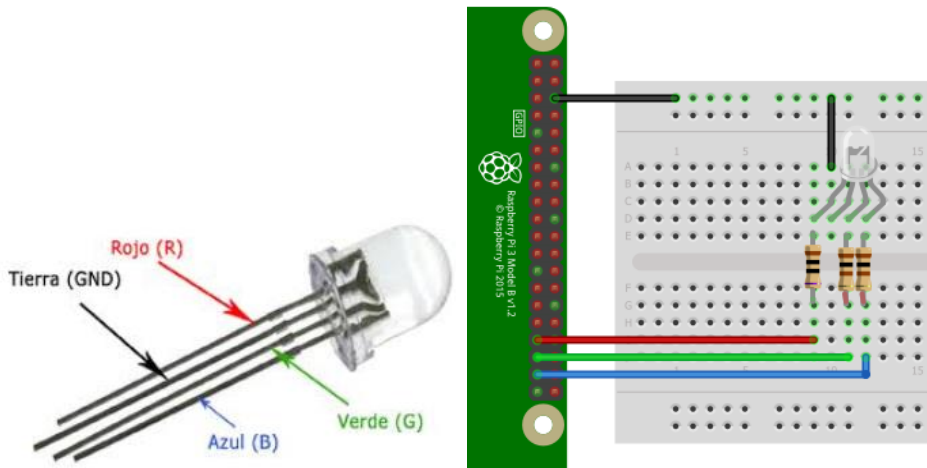


Práctica: **Control de un LED RGB**

En esta práctica vamos a realiza el control de un LED tricolor. Realmente se trata de tres LEDs dentro de la misma cápsula. Uno es rojo, otro verde y otro azul, de manera que, combinando la activación de cada uno, podemos conseguir distintas tonalidades. Lo ideal sería conectarlos a salidas PWM para controlar la intensidad

de cada componente. Por desgracia, la Raspberry Pi solo tiene 2 salidas PWM. Por esta razón y para simplificar la programación vamos a utilizar salidas GPIO binarias. Al disponer de 3 bits para su control vamos a poder generar 7 colores diferentes, más el estado apagado.

Realiza un programa que haga pasar el LED por cada uno de estos 8 estados a intervalos de un segundo y lo repita de forma cíclica. Utiliza las salidas BCM13, BCM19 y BCM26. Puedes utilizar el esquema que se muestra a continuación. En principio las tres resistencias pueden ser de 100Ω. No obstante, dependiendo del LED, es posible que tengas que ajustar algún valor. Por ejemplo, en el esquema se ha utilizado una resistencia de 10Ω para el componente rojo.

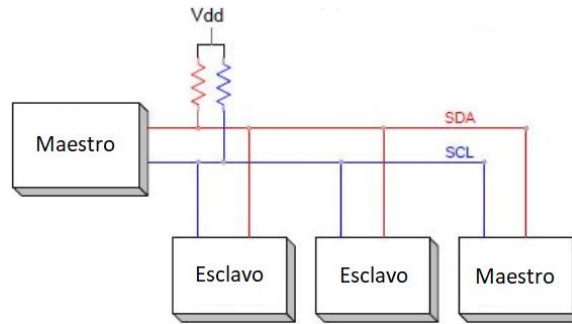


NOTA: Si no dispones de un LED RGB, puedes realizar la práctica usando tres LEDs diferentes. Preferiblemente de colores rojo, verde y azul.

1.6.3. Bus series I²C

El bus I²C³ permite interconectar varios dispositivos utilizando una modalidad de transmisión serie y síncrona. Su nombre viene del inglés, *Inter-Integrated Circuit*, lo que nos indica que ha sido diseñado para la interconexión de circuitos integrados. Fue desarrollado por Philips a principios de los 80, para reducir el número de pines en los chips. Como vamos a ver I²C permite controlar múltiples dispositivos utilizando solo 2 pines. A diferencia de otros interfaces de comunicación, como UART o USB que solo permite conectar dos dispositivos entre sí, I²C permite conectar varios dispositivos utilizando una topología en bus:

³ <https://en.wikipedia.org/wiki/I%C2%B2C>

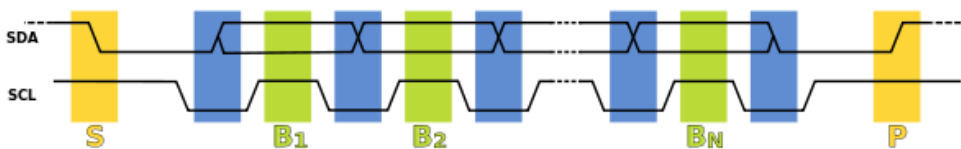


Cada dispositivo puede tomar el rol de maestro o el de esclavo. Una transferencia de datos siempre la inicia un maestro, y es un esclavo el que contesta. Lo más habitual es disponer de un solo maestro, pero existe un modo multimetro.

Se utiliza una topología en bus con dos líneas: SDA (Serial Data) para transmitir los datos en serie y SCL (Serial CLock) con una señal que marca el comienzo de cada bit. Al utilizar esta señal de reloj se dice que es una transmisión síncrona. En el bus se añaden dos resistencias de pull-up de forma que, si ningún dispositivo está utilizando el bus, la línea estará a nivel alto, es decir a 1.

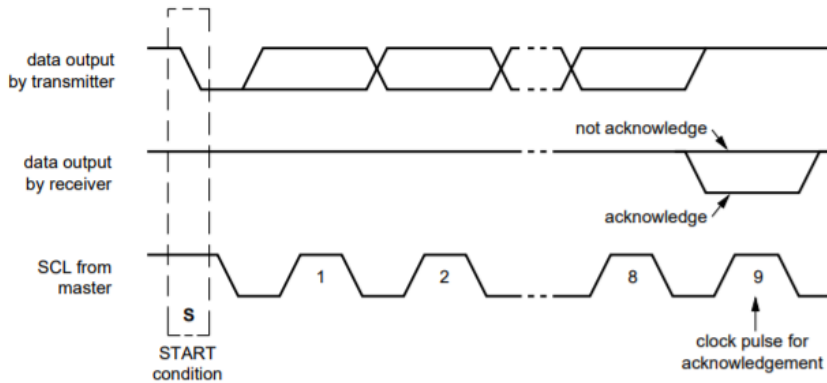
La señal de reloj siempre es introducida por el maestro, por lo que la velocidad de transmisión puede variar según marque el maestro. Existen varios valores preestablecidos, 0,1 0,4 1,0 3,4 y 5,0 Mbits/s. La unidad de datos transferida siempre es un byte.

El primer byte enviado por el maestro siempre es la dirección del esclavo en 7 bits, seguido de un 1 si quiere leer del esclavo o un 0 si quiere escribir. A este bit se le llama R/W. De los 7 bits de la dirección, los 4 más significativos son predeterminados por el fabricante y los tres últimos pueden ser fijados por tres (jumpers) que podemos configurar en el circuito. Esto nos permite introducir hasta 8 (2^3) circuitos iguales en el bus.



Si se realiza un flanco de bajada en la línea de datos, estando la línea de reloj a 1, significa que se inicia la transmisión (S: bit de arranque). Cuando el reloj baje, se pondrá el primer bit en la línea, cuando el reloj suba el bit podrá leerse. hasta que el reloj no vuelva a bajar el valor del bit ha de permanecer estable (B: bit de datos). Este proceso se repite hasta transmitir los N bits. Un blanco de subida, mientras el reloj está a 1, significa que termina la transmisión (P: bit de parada).

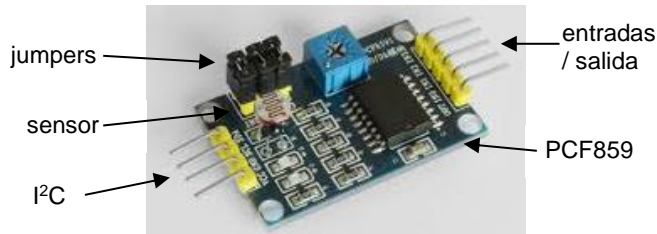
Tras la transmisión de cada byte se espera un bit de reconocimiento por parte del receptor (es decir, tras recibir 8 bits el receptor transmitirá un bit). Si el bit es 0, el reconocimiento es positivo y 1 en caso contrario.



Para indicar el final de una transmisión, el último byte leído es reconocido por el maestro como un no reconocimiento (Not ACKnowledge). Una transmisión es finalizada por la señal de parada.

1.6.3.1. El convertor A/D y D/A PCF8591

El PCF8591 es un chip que implementa un convertor de entradas analógicas en digitales de cuatro canales y un convertor digital/analógico de una salida. La resolución es de 8 bits. Este chip se conecta a un dispositivo maestro mediante el bus I²C. La velocidad máxima de conversión viene dada por la velocidad máxima del bus I²C.



El direccionamiento es de 7 bits, siendo los cuatro más significativos 1001 y los tres últimos configurables mediante jumpers. Si dejas los tres puestos estos tres bits serán 000. El primer byte enviado por el maestro enviando la dirección del dispositivo, seguido del bit de lectura/escritura.

7	6	5	4	3	2	1	0
MSB							LSB
1	0	0	1	A2	A1	A0	R/W

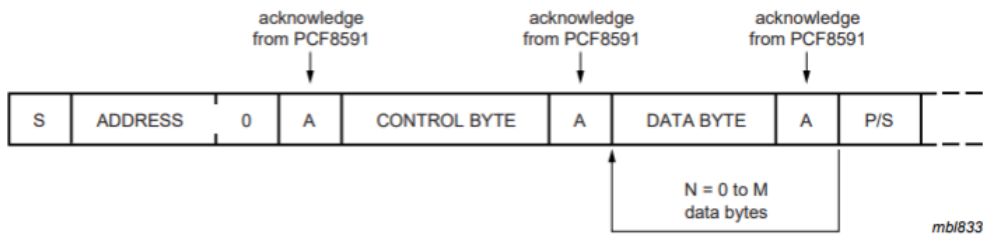
Conversión D/A

El maestro envía la dirección del PCF8591, seguida de un 0 en el bit R/W. El segundo byte que se envía es el byte de control, con los siguientes bits:

bit	valor	significado
7	0	valor fijo
6	0/1	1 activamos la salida analógica

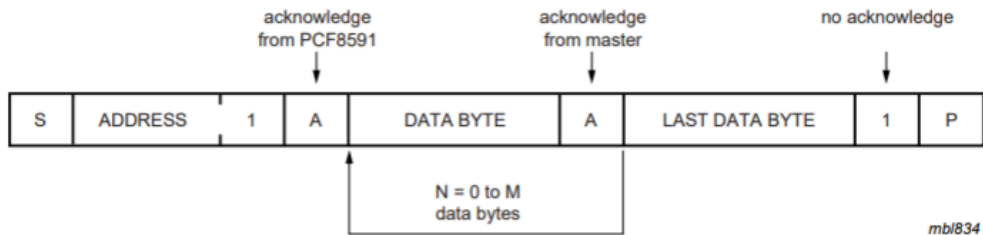
5,4	00	4 entradas analógicas AIN0-AIN3 respecto a GND
	01	3 entradas analógicas respecto a AIN3
	10	AIN0 y AIN1 respecto a GND, AIN2 respecto a AIN3
	11	AIN0 respecto a AIN1, AIN2 respecto a AIN3
3	0	valor fijo
2	0/1	1 activamos auto incremento
1,0	00	canal 0
	01	canal 1
	10	canal 2
	11	canal 3

El tercer byte enviado es el valor digital a convertir. Podemos continuar la transmisión de bytes, de forma que el valor de salida será actualizado, hasta que el maestro introduzca un bit de parada.



Conversión A/D

El maestro envía la dirección del PCF8591, seguida de un 1 en el bit R/W. El PCF8591 envía la conversión digital de la entrada analógica. El canal y si se utiliza un voltaje diferencial se configura en el byte de control. El valor de la conversión será vuelto a mandar hasta que el maestro mande un reconocimiento negativo y un bit de parada. Si el modo activamos auto incremento está activo, se enviarán los diferentes canales uno tras otro, de forma cíclica.



Para una descripción más detallada consulta su data sheet⁴:

1.6.3.2. Utilización del bus I²C desde Android Things

Vemos como configurar el bus I²C desde el API de Android Things:

⁴ <https://www.nxp.com/docs/en/data-sheet/PCF8591.pdf>

El primer paso va a ser conocer el nombre del puerto al que te quieres conectar. Para conocer los nombres de los dispositivos conectados utiliza el siguiente código:

```
PeripheralManager manager = PeripheralManager.getInstance();
List<String> listaDispositivos = manager.getI2cBusList();
```

En una Raspberry Pi 3, posiblemente obtendrás el nombre: **I2C1**. El siguiente código muestra un ejemplo de escritura y lectura, aplicado al chip PCF8591:

```
private static final byte ACTIVA_SALIDA = 0x40; // 0100 00 00
private static final byte AUTOINCREMENTO = 0x04; // 0000 01 00
private static final byte ENTRADA_0 = 0x00; // 0000 00 00
private static final byte ENTRADA_1 = 0x01; // 0000 00 01
private static final byte ENTRADA_2 = 0x02; // 0000 00 10
private static final byte ENTRADA_3 = 0x03; // 0000 00 11
private static final String IN_I2C_NOMBRE = "I2C1"; // Puerto de entrada
private static final int IN_I2C_DIRECCION = 0x48; // Dirección de entrada
private I2cDevice i2c;

...
try {
    i2c = manager.openI2cDevice(IN_I2C_NOMBRE, IN_I2C_DIRECCION);

    byte[] config = new byte[2];
    config[0] = (byte) ACTIVA_SALIDA + ENTRADA_0; // byte de control
    config[1] = (byte) 0x80; // valor de salida (128/255)
    i2c.write(config, config.length); // escribimos 2 bytes

    byte[] buffer = new byte[5];
    i2c.read(buffer, buffer.length); // Leemos 5 bytes
    String s = "";
    for (int i=0; i<buffer.length; i++) {
        s += " byte "+i+": " + (buffer[i]&0xFF);
    }
    Log.d(TAG, s); // mostramos salida

    i2c.close(); // cerramos i2c
    i2c = null; // liberamos memoria
} catch (IOException e) {
    Log.e(TAG, "Error en al acceder a dispositivo I2C", e);
}
```

En el código anterior comenzamos abriendo el dispositivo I2C. De los dos disponibles seleccionamos el número 1 ("I2C1"). Además, indicamos que entre los diferentes esclavos que puede estar conectados al bus nos interesa el que tiene dirección hexadecimal 48 (en binario 1001000).

Luego creamos un array con 2 bytes que serán enviados por el bus. El primero es el byte de control, donde indicamos que queremos activar la salida OUT y además realizar una lectura de la línea IN0. El segundo byte es el vamos que vamos a convertir de digital a analógico. Se indica el valor hexadecimal 80, que equivale a 128 en decimal. El valor máximo de un byte es 255, por lo que estaríamos en el valor medio. Si alimentamos el conversor a 5V, el valor esperado a la salida será 2,5V.

El tercer paso consiste en leer la entrada analógica, para lo que comenzamos creando un array de 5 bytes. Luego los bytes son mostrados en el log. El primer byte leído es posible que siempre sea 0. Desconocemos el significado de este primer byte. Los cuatro siguientes han de ser cuatro lecturas consecutivas obtenidas de la entrada 0. Estos valores serán 0 si en la entrada está a 0V y 255 si en la entrada está al valor de alimentación (5V). El resto de los valores se obtienen de forma lineal.

El código concluye cerrando el dispositivo y haciendo el procesado de errores.

Este código hace una escritura/lectura en crudo (raw), adaptada al protocolo de comunicación definido para PCF8591 (ver sección anterior). Sin embargo, otros dispositivos I2C, siguen un protocolo estándar conocido como System Management Bus ([SMBus](#)). En este protocolo se definen una serie de registros que pueden ser de lectura o escritura y se les asigna una dirección. Para más información sobre cómo utilizarlo consultar la documentación oficial⁵.



Ejercicio: Una entrada/salida I2C en Android Things

Material necesario:

- una placa con el chip PCF8591.
- un LED para mostrar la salida y una resistencia para ajustar su intensidad.
- (opcional) potenciómetro o fotoresistencia para obtener una entrada analógica. Si has comprado el Kit de 16 sensores para Arduino puedes utilizar el módulo de sensor fotoresistor. Este sensor dispone de dos salidas: La analógica que nos ofrece un valor entre 0V y el nivel de alimentación inversamente proporcional al nivel de luz detectado. También tiene una salida digital que se activa al superar un nivel determinado. Este nivel puede ser configurado por medio de un pequeño potenciómetro.



1. Realiza las conexiones del bus I²C tal y como se muestra en la figura. En este caso lo alimentamos a 5V, aunque también se podría alimentar a 3,3V.

⁵ <https://developer.android.com/things/sdk/pio/i2c.html>

2. Crea un nuevo proyecto con los siguientes datos:

☐ Phone and Tablet

Minimum SDK: API 27 Android 8.1 (Oreo)

Activity Name: MainActivity

- Generate a UI layout File

- Añade el siguiente código mostrado anteriormente en el ejemplo anterior, dentro de `onCreate()`.

```
<uses-permission
    android:name="com.google.android.things.permission.USE_PERIPHERAL_IO"/>
```

- Verifica que los tres jumpers de la placa PCF8591 coinciden con la dirección del bus I²C.

6. Para verificar la salida analógica, puedes utilizar un voltímetro midiendo entre el pin OUT y GND. Como hemos indicado 255, el valor ha de ser de 5V. También puedes utilizar un LED (recuerda utilizar la resistencia adecuada). El LED ha de brillar a máxima potencia.

7. Modifica el valor de salida a 128, el valor leído ha de ser de 2,5V y el LED ha de perder brillo.

8. Modifica el valor de salida a 0, el valor leído ha de ser de 0V y el LED ha de apagarse.
9. Para verificar las entradas conecta IN0 a GND. Los cuatro valores leídos han de ser 0. Conecta IN0 a 3,3V. Los cuatro valores leídos han de ser 255*3,3/5. Conecta IN0 a 5V. Los cuatro valores leídos han de ser 255.
10. Si dispones de un potenciómetro o una fotorresistencia, utilízala como entrada.
11. Trata de configurar el modo autoincremental para que los valores leídos correspondan a entradas IN0 a IN3.

Lista de dispositivos I2C: [I2C1]
 byte 0: 0 byte 1: 0 byte 2: 168 byte 3: 255 byte 4: 44

1.6.3.3. Utilización del chip PCF8591 por medio de un driver

Como acabamos de ver utilizar un chip puede ser bastante complejo. Este puede tener varios modos de funcionamiento o complicados protocolos de comunicación. Comprender las especificaciones que nos ofrecen en el data sheet, puede llevarnos mucho tiempo.

Afortunadamente este trabajo suele estar ya resuelto, de forma que no tenemos más que instalarnos el driver del dispositivo. Este se encargará de realizar las tareas complejas, ofreciéndonos un interfaz mucho más sencillo.

Por ejemplo, para el chip PCF8591 podemos encontrar el siguiente driver:

<https://github.com/davemckelvie/things-drivers/tree/master/pcf8591>



Ejercicio: Usar un driver para PCF8591

NOTA: El material necesario y las conexiones coinciden con el ejercicio anterior.

1. Puedes crear un nuevo proyecto o partir del ejercicio anterior comentando el código.
2. Añade la siguiente dependencia en build.gradle (app):

```
dependencies {
    ...
    implementation 'nz.geek.android.things:things-drivers:1.8.0'
}
```

3. Añade los siguientes permisos en AndroidManifest:

```
<uses-permission
    android:name="com.google.android.things.permission.MANAGE_INPUT_DRIVERS"/>
<uses-permission
    android:name="com.google.android.things.permission.USE_PERIPHERAL_IO"/>
```

4. Declara las siguientes variables:

```
private I2cAdc adc;
private Handler handler = new Handler();
private Runnable runnable = new UpdateRunner();
```

5. En el método onCreate() añade:

```
I2cAdc.I2cAdcBuilder builder = I2cAdc.builder();
adc = builder.address(0).fourSingleEnded().withConversionRate(100).build();
adc.startConversions();
handler.post(runnable);
```

Se crea un builder que nos permitirá configurar el conversor AD. Se indica la dirección, pero solo la indicada en los jumpers (para 0 los tres ha de estar insertados). El modo de funcionamiento (lectura de los cuatro canales) y cada cuanto queremos una conversión.

Al llamar al método `startConversions()` se inicia la lectura de datos de forma continua. Para evitar fluctuaciones en los datos se utiliza la siguiente ecuación:

$$\text{valor de salida} = (\text{salida anterior} + \text{nueva lectura}) / 2$$

6. Añade el siguiente código:

```
private class UpdateRunner implements Runnable {
    @Override public void run() {
        String s = "";
        for (int i=0; i<=3; i++) {
            s += " canal "+i+": "+adc.readChannel(i);
        }
        Log.d(TAG, s);
        handler.postDelayed(this, 1000);
    }
}
```

Se muestra los valores obtenidos en cada canal, en periodo de 1 segundo.

El driver que estamos utilizando tiene el inconveniente de no permitir la salida de datos utilizando el conversor D/A.

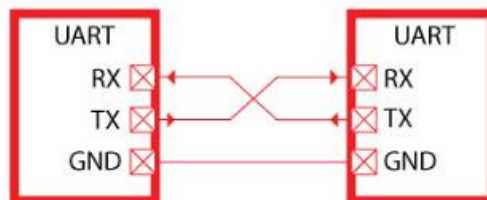
1.6.4. Entradas / salidas series SPI

<https://developer.android.com/things/sdk/pio/spi.html>

<http://android.geek.nz/measuring-analog-values-with-android-things/>

1.6.5. Entradas / salidas series UART

UART (Universal Asynchronous Receiver-Transmitter) es una interfaz de comunicación serie con la que podemos conectar gran variedad de dispositivos como GPS o pantallas LCD. Se conoce como un interfaz asíncrono dado que no utiliza una señal de reloj.



Como se muestra en el esquema anterior utiliza tres cables para la conexión. Los cables de transmisión (TX) y recepción han de estar cruzados entre los dos

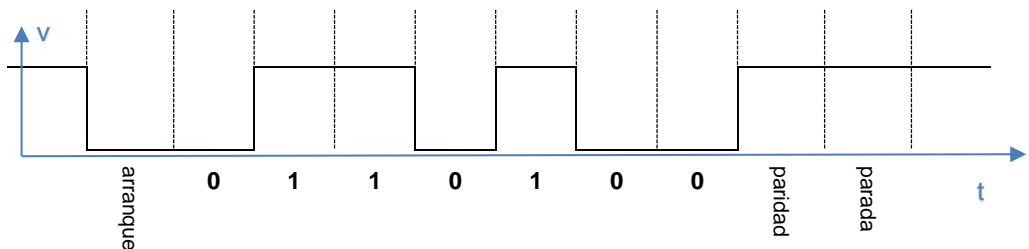
dispositivos. De este esquema podemos extraer que la transmisión es full dúplex (podemos transmitir y recibir a la vez) y que solo podemos interconectar dos dispositivos.

Uno de los mayores inconvenientes de la comunicación por UART es que emisor y receptor han de ponerse de acuerdo en varios parámetros de configuración. Para entender, estos parámetros veamos cómo se transmite la información.

La línea ha de permanecer a nivel alto mientras no se transmite información. Cada vez que se quiere transmitir una palabra, se transmite el **bit de arranque**, consiste en poner la línea a nivel bajo durante un periodo de tiempo, de la duración de un bit. Luego se transmitirían los **bits de datos**, pudiéndose configurar que estos sean entre 5 y 8. Cada bit se representa como un nivel alto para el "1" y nivel bajo para el "0", durante un periodo de bit. Los bits son transmitidos de derecha a izquierda, es decir primero el menos significativo. A continuación, se puede añadir un **bit de paridad**, que puede ser: par (la suma de "1" transmitidos es par), impar (la suma de "1" transmitidos es impar) o ninguna (no se añade este bit). Finalmente, se transmite el **bit de parada**. Consiste en dejar la línea a nivel alto un mínimo de uno o dos periodos de bit, según hayamos configurado.

Tras el bit de parada se podría transmitir una nueva palabra, comenzando por su bit de arranque. Pero si no disponemos de más datos la línea quedaría a nivel alto un tiempo indefinido.

En la siguiente gráfica se muestra la transmisión de la palabra 0010110, utilizando 7 bits de datos, 1 bit de paridad par y 1 bit de parada.



Si el receptor no tiene estos parámetros correctamente configurados, la transmisión no podrá realizarse. Otro aspecto de vital importancia es que emisor y receptor han de ponerse de acuerdo en el tiempo de bit. Cualquier pequeña diferencia entre los relojes de cada extremo, provocará un error en la transmisión. Este tiempo se indica en baudios (bits por segundo) siendo los valores más habituales 300, 1200, 4800, 9600, ..., 115200, ..., 1M, 2M baudios.

Vemos como configurar los puertos UART desde el API de Android Things:

El primer paso va a ser conocer el nombre del puerto al que te quieres conectar. Para conocer los nombres de los dispositivos conectados utiliza el siguiente código:

```
PeripheralManager manager = PeripheralManager.getInstance();
List<String> listaDispositivos = manager.getUartDeviceList();
```

En una Raspberry Pi 3, obtendremos los siguientes nombres: **MINIUART** y **UART0**. Nuevos dispositivos UART pueden ser conectados a través de un puerto USB. Por lo que la lista anterior puede aumentar si el usuario ha conectado nuevos dispositivos.

Para abrir el UART y configurarlo utilizaremos:

```
UartDevice uart;
...
try {
    uart = manager.openUartDevice("UART0");
    uart.setBaudrate(115200);
    uart.setDataSize(8);
    uart.setParity(UartDevice.PARITY_NONE);
    uart.setStopBits(1);
} catch (IOException e) {
    Log.w(TAG, "Error iniciando UART", e);
}
```

Para cerrarlo:

```
uart.close();
```

Para escribir bytes utilizaríamos:

```
byte[] buffer = {...};
int bytesEscritos = uart.write(buffer, buffer.length());
```

Si queremos escribir los caracteres de un String:

```
int bytesEscritos = uart.write(s.getBytes(), s.length());
```

Los datos recibidos son almacenados internamente en una memoria FIFO. Podremos extraer estos datos utilizando:

```
int bytesLeídos = uart.read(buffer, buffer.length);
```

Los datos son eliminados de la memoria de lectura y copiados a **buffer**. Pero si los datos superan al valor indicado en el segundo parámetro, solo se extraerán estos bytes. Para asegurarnos que todos los datos son extraídos independientemente del tamaño de buffer, podemos utilizar este código.

```
byte[] buffer = new byte[16]; // Máximo de datos leídos cada vez 16
do {
    bytesLeídos = uart.read(buffer, buffer.length);
    // Procesamos un máximo de 16 bytes
} while(bytesLeídos >0);
```

También podemos programar un escuchador de evento que se active cada vez que lleguen nuevos datos:

```
uart.registerUartDeviceCallback(
    new UartDeviceCallback() {
        @Override public boolean onUartDeviceDataAvailable(UartDevice uart){
            try {
                // Usar código anterior para leer bytes
            }
        }
    }
```

```

    } catch (IOException e) {
        Log.w(TAG, "Error al leer de UART", e);
    }
    return true; // Continue listening for more interrupts
}

@Override public void onUartDeviceError(UartDevice uart, int error){
    Log.w(TAG, uart + ": Error " + error);
}
}
);

```

En el ejercicio *Usar Arduino como esclavo a través de UART* se muestra un ejemplo del uso de este API.

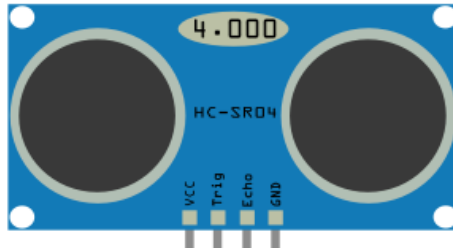
1.6.6. Medidor ultrasónico de distancia.

En los siguientes apartados vamos a hacer uso del sensor ultrasónico HC-SR04 para medir distancias, por lo que comenzamos describiendo su funcionamiento. Se basa en el principio del sonar para determinar la distancia de un objeto, de forma similar a como lo hacen los murciélagos o un submarino. Es decir, se emite un pulso de ultrasonidos y se mide el tiempo del rebote. Conociendo la velocidad de propagación del sonido por el aire y sabiendo que el sonido tiene que ir y volver, calcular la distancia a partir de este tiempo es sencillo:



El sensor HC-SR04 tiene un rango de funcionamiento de 2 a 400 cm. Puede hacer medidas en un ángulo de 30°. Su funcionamiento no se ve afectado por la luz solar o el color del material, aunque los materiales como la tela, pueden ser difíciles de detectar.

Dispone de cuatro conectores: **VCC** que ha de conectarse a 5V y **GND** a masa. **Trig** es una entrada que ha de activarse un mínimo de 10µs para solicitar una medida. Entonces el sensor emitirá una ráfaga de 8 pulsos a 40KHz. Activará la salida **Echo** todo el tiempo que tarda en llegar el rebote de esta señal. Para obtener la distancia en cm, has de dividir el ancho del pulso obtenido en Echo entre 58.



Puedes encontrar el datasheet del sensor en el siguiente link⁶.



Ejercicio: Medidor ultrasónico de distancia con Android Things

NOTA: Este ejercicio está basado en el siguiente tutorial de [Daniel Dallos](#).

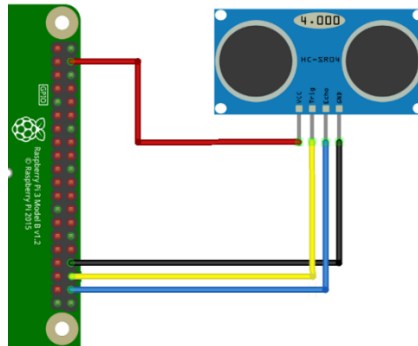
Material necesario:

- Sensor de distancia HC-SR04.

1. Crea un nuevo proyecto para Android Things, sin layout, de forma similar a como se ha hecho en los ejercicios anteriores.

NOTA: Si lo prefieres puedes realizar este ejercicio dentro del mismo proyecto que el anterior, para facilitar la entrega de la tarea.

2. Conecta el sensor de distancia a la Raspberry Pi. El siguiente esquema te muestra cómo hacerlo:



3. Añade las siguientes variables:

```
private static final String ECHO_PIN_NAME = "BCM20";
private static final String TRIGGER_PIN_NAME = "BCM16"; //antes 21
private static final int INTERVALO_ENTRE_LLECTURAS = 3000;
private Gpio mEcho;
private Gpio mTrigger;
```

⁶https://docs.google.com/document/d/1Y-yZnNhMYy7rwhAgyL_pfa39RsB-x2qR4vP8saG73rE

4. Inicializa los puertos los puertos GPIO en `onCreate()`:
5. Añade el siguiente método:

```
int hazAlgo;

protected double leerDistancia() throws IOException, InterruptedException{
    mTrigger.setValue(false);
    Thread.sleep(0, 2000); // 2 mseg
    mTrigger.setValue(true);
    Thread.sleep(0, 10000); //10 msec
    mTrigger.setValue(false);
    while (mEcho.getValue() == false) {
        hazAlgo = 0;
    }
    long tiempoIni = System.nanoTime();
    while (mEcho.getValue() == true) {
        hazAlgo = 1;
    }
    long tiempoFin = System.nanoTime();
    long anchoPulso = tiempoFin - tiempoIni;
    double distancia = (anchoPulso / 1000.0) / 58.23; //cm
    Log.i(TAG, "distancia (Android Things): " + distancia + " cm");
    return distancia;
}
```

6. Añade el código necesario para que el método sea llamado cada 3 segundos.
7. Ejecuta el programa y haz una estimación de la precisión conseguida.
8. ¿En algún momento el programa deja de funcionar? ¿Cuál puede ser la causa? ¿Cómo podrías resolver el problema?



Preguntas de repaso: [Entradas / Salidas en Android Things](#)

1.7. Usar un microcontrolador Arduino como esclavo

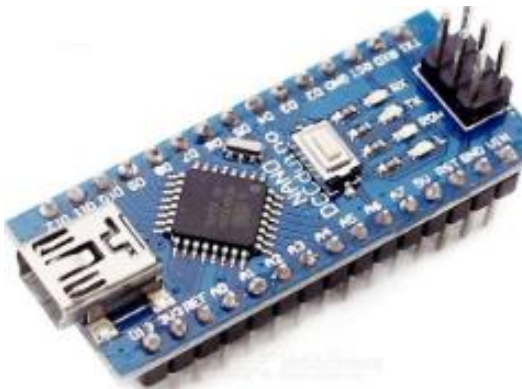
Si has realizado el ejercicio anterior, habrás comprobado que la Raspberry Pi no es el dispositivo más adecuado para interaccionar con los sensores. Esto es debido a varias causas:

- Al estar basado en un sistema operativo multiproceso, como Linux, no podemos asegurar que tenemos el procesador 100% disponible para nosotros.
- Las entradas GPIO en Android Things son especialmente lentas. Si quisiéramos generar una señal binaria activando y desactivando una salida

GPIO, la frecuencia máxima que podríamos alcanzar es de 3KHz⁷. El problema es que muchos sensores necesitan una señal de más frecuencia, por lo que no vamos a poder controlarlos directamente desde Android Things. Podrías preguntarte, porque ocurre esto si los procesadores con los que estamos trabajando son muy rápidos. La razón estaría en que los desarrolladores de Google a la hora de desarrollar este API, han utilizado el método SYSFS, primado la seguridad frente a la velocidad.

- Otro inconveniente lo encontramos en la falta de entradas/salidas GPIO analógicas. Este problema se puede paliar utilizando un conversor A/D.
- La mayoría de sensores han sido diseñados para ser usados desde un microcontrolador. En sus especificaciones suelen incluirse fragmentos de código para descubrir su utilización en entorno Arduino. En algunos casos incluso se incluye una librería para controlarlos.

Todos estos inconvenientes no aparecen cuando estamos usando un microcontrolador. Al tener un único hilo de ejecución, no tenemos el problema de ser interrumpidos en mitad de un proceso. Pueden trabajar generando una señal binaria en una salida GPIO de hasta 8MHz. Disponen de entradas analógicas. Al tener un propósito muy concreto, normalmente controlar un sensor, suelen ser sencillos de programar, y por lo tanto más fiables. El consumo es muy bajo. Algunos incorporan mecanismos para entrar en suspensión cuando no son requeridos y volver a estar operativos en milisegundos. Esto permite que puedan tener una autonomía de varios años, alimentados con una pequeña batería. Finalmente, su precio es muy reducido. A continuación, se muestran dos microcontroladores, junto con su precio, que utilizaremos en los siguientes ejercicios:



Arduino Nano ([3,22 €](#))



ATTINY85 con USB ([1,34 €](#))

A la izquierda tenemos un Arduino Nano que incorpora un microcontrolador ATmega328 (mismo que utiliza Arduino UNO). Este chip tiene 14 entradas/salidas (6 son PWM y 6 son conversores A/D con 10 bits de resolución), 1 KB de EEPROM,

⁷ <https://stackoverflow.com/questions/41727580>

2 KB de SRAM, 32 KB de memoria flash, frecuencia 16Mhz. El microcontrolador se monta sobre una placa que puede ser fácilmente insertada sobre una placa de prototipos. Además, incorpora un USB que facilita su programación, o como veremos más adelante la conexión con la Raspberry Pi.

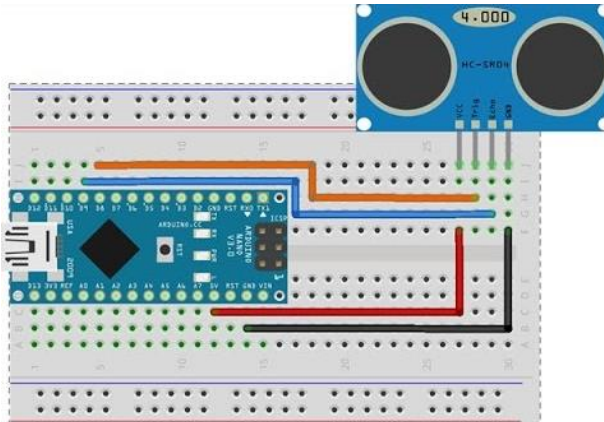
A la derecha tenemos el microcontrolador ATTINY85 montado en una placa que nos facilita la conexión directa a un procesador con entrada USB. Por ejemplo, en la Raspberry Pi podríamos utilizar una de las 4 disponibles. También podemos conectarlo a un USB de nuestro ordenador para programarlo utilizando la plataforma Arduino. En tamaño del microcontrolador es muy pequeño, 0,9 x 0,7 mm y tiene solo 8 pines (chip de la izquierda). Sus prestaciones también son reducidas. 5 entradas/salidas (2 son PWM), 512 bytes de EEPROM, 512 bytes de SRAM, frecuencia 20Mhz.



Ejercicio: Medidor ultrasónico de distancia con Arduino

Material necesario:

- Microcontrolador compatible Arduino (hemos utilizado Arduino Nano).
 - Cable con conector de USB a Mini USB, conector macho tipo "A" a macho mini USB tipo "B" 5 pines.
 - Sensor de distancia HC-SR04.
1. Monta el sensor de distancia y el Arduino Nano sobre la placa de prototipos, tal y como se muestra a continuación:



Ha de conectar GND, 5V, D8 y D9 en Arduino con GND, VCC, Trig y Echo en el sensor.

2. Accede a la página <https://www.arduino.cc/en/Main/Software> y busca la sección *Download the Arduino IDE*. Selecciona la descarga adecuada según tu sistema operativo e instálalo.
3. Abre el entorno y copia el siguiente código en el editor:

```

const int EchoPin = 9;
const int TriggerPin = 8;

void setup() {
  Serial.begin(115200);
  pinMode(TriggerPin, OUTPUT);
  pinMode(EchoPin, INPUT);
}

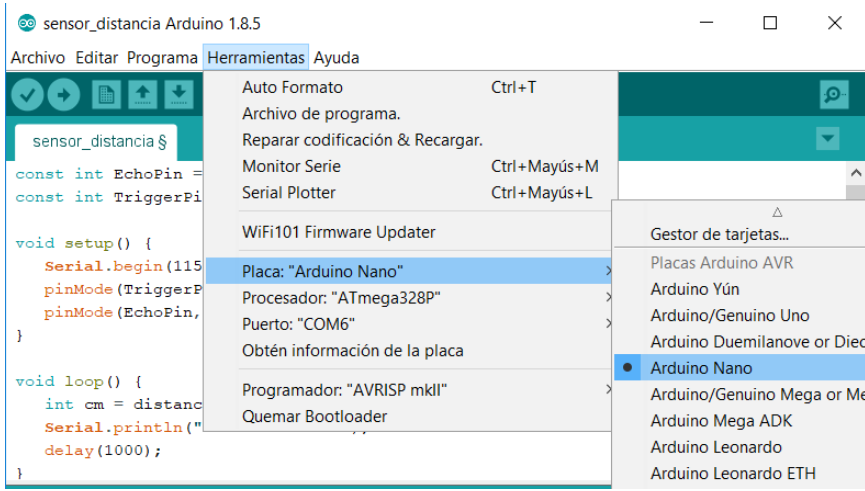
void loop() {
  Serial.print("Distancia: ");
  Serial.println(distancia(TriggerPin, EchoPin));
  delay(1000);
}

int distancia(int TriggerPin, int EchoPin) {
  long duracion, distanciaCm;
  digitalWrite(TriggerPin, LOW); //nos aseguramos señal baja al principio
  delayMicroseconds(4);
  digitalWrite(TriggerPin, HIGH); //generamos pulso de 10us
  delayMicroseconds(10);
  digitalWrite(TriggerPin, LOW);
  duracion = pulseIn(EchoPin, HIGH); //medimos el tiempo del pulso
  distanciaCm = duracion * 10 / 292 / 2; //convertimos a distancia
  return distanciaCm;
}

```

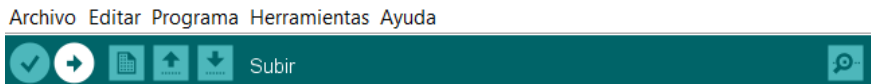
El lenguaje de programación que utiliza Arduino está basado en C/C++. Siempre han de existir dos métodos. `setup()` que es ejecutado al arrancar el dispositivo para realizar las configuraciones iniciales. En el código se inicializa el puerto serie, con una determinada velocidad, para utilizarlo como salida de consola. Además, se inicializa los dos puertos GPIO que vamos a utilizar. El método `loop()` será ejecutado de forma continua, una tras otra. En el código, mostramos por el puerto serie "Distancia: " y luego mostramos el resultado obtenido por la función `distancia`. Finalmente se espera un segundo y se repite el proceso. El funcionamiento de la función `distancia()` resulta sencillo de entender, una vez leído el apartado anterior.

4. Conecta un cable USB desde Arduino a tu ordenador.
5. Selecciona la opción Herramientas/Puerto/COMX. Donde X es el número de puerto donde has conectado el Arduino. Si hay varios donde escoger, desconecta el cable y mira qué número de puerto desaparece.
6. Selecciona la opción Herramientas/Placa/Arduino Nano. O la placa que estés utilizando:

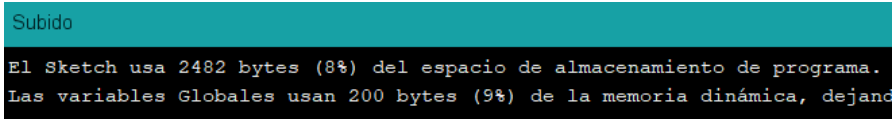


7. Selecciona la opción *Herramientas/Procesador/ATmega328P (Old Bootloader)*

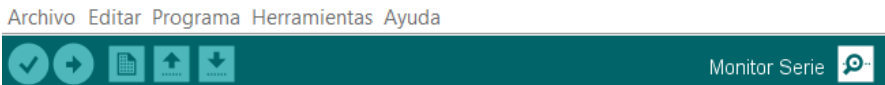
8. Pulsa el botón *Subir*.



9. Si todo es correcto en la parte inferior ha de aparecer:

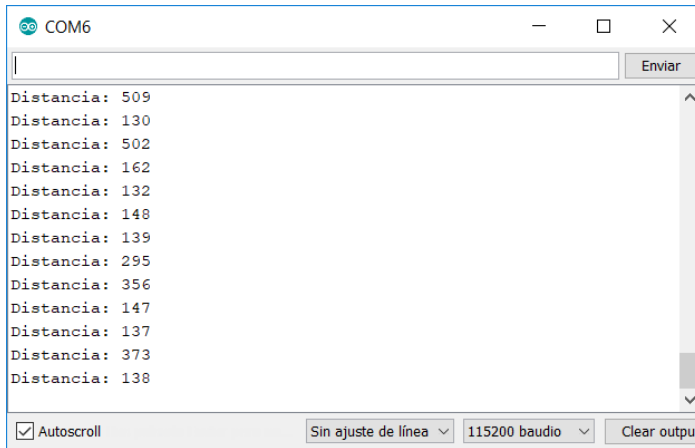


10. Pulsa el botón *Monitor Serie*:



Se abrirá una ventana que me permite comunicarme a través del puerto serie con Arduino.

11. Selecciona en el desplegable de la parte inferior 115200 baudios. La salida se actualizará cada segundo mostrando la distancia entre el sensor y el objeto más cercano:



Una vez descrito como podemos controlar el sensor de distancia con Arduino vamos a ver podemos utilizarlo desde Android Things para acceder al sensor a través del microcontrolador.



Ejercicio: Procesar comandos en Arduino por el puerto serie

1. Modifica el método `loop()` del ejercicio anterior por el siguiente

```
void loop() {
  if (Serial.available() > 0) {
    char command = (char) Serial.read();
    switch (command) {
      case 'H':
        Serial.println("Hola Mundo");
        break;
      case 'D':
        Serial.println(distancia(TriggerPin, EchoPin));
        break;
    }
  }
  Serial.flush();
}
```

Este código implementa un sistema de comandos con el que podremos solicitar al microcontrolador la lectura de diferentes sensores, enviando un carácter determinado.

2. Sube el nuevo código al dispositivo y abre el *Monitor Serie*.
3. Escribe el carácter **H** y pulsa en *Enviar*. Ha de aparecer el texto **Hola mundo**.
4. Escribe el carácter **D** y pulsa en *Enviar*. Se mostrará el valor del sensor.



Ejercicio: Usar Arduino como esclavo a través de UART

1. Añade la siguiente clase en el proyecto:

```
public class ArduinoUart {

    private UartDevice uart;

    public ArduinoUart(String nombre, int baudios) {
        try {
            uart = PeripheralManager.getInstance().openUartDevice(nombre);
            uart.setBaudrate(baudios);
            uart.setDataSize(8);
            uart.setParity(UartDevice.PARITY_NONE);
            uart.setStopBits(1);
        } catch (IOException e) {
            Log.w(TAG, "Error iniciando UART", e);
        }
    }

    public void escribir(String s) {
        try {
            int escritos = uart.write(s.getBytes(), s.length());
            Log.d(TAG, escritos+" bytes escritos en UART");
        } catch (IOException e) {
            Log.w(TAG, "Error al escribir en UART", e);
        }
    }

    public String leer() {
        String s = "";
        int len;
        final int maxCount = 8; // Máximo de datos leídos cada vez
        byte[] buffer = new byte[maxCount];
        try {
            do {
                len = uart.read(buffer, buffer.length);
                for (int i=0; i<len; i++) {
                    s += (char)buffer[i];
                }
            } while(len>0);
        } catch (IOException e) {
            Log.w(TAG, "Error al leer de UART", e);
        }
        return s;
    }

    public void cerrar() {
        if (uart != null) {
            try {
                uart.close();
            }
        }
    }
}
```

```

        uart = null;
    } catch (IOException e) {
        Log.w(TAG, "Error cerrando UART", e);
    }
}

static public List<String> disponibles() {
    return PeripheralManager.getInstance().getUartDeviceList();
}
}

```

2. Añade la siguiente código en el método `onCreate()`:

```

Log.i(TAG, "Lista de UART disponibles: " + ArduinoUart.disponibles());
ArduinoUart uart = new ArduinoUart("UART0", 115200);
Log.d(TAG, "Mandado a Arduino: H");
uart.escribir("H");
try {
    Thread.sleep(5000);
} catch (InterruptedException e) {
    Log.w(TAG, "Error en sleep()", e);
}
String s = uart.leer();
Log.d(TAG, "Recibido de Arduino: "+s);

```

3. Puntea los pines TX (8) y RX (10). De esta forma todo lo transmitido será directamente recibido.

NOTA: En Arduino la conexión UART de estos dos pines es compartida por la que se usa en el puerto USB para conectarlo al PC. En caso de estar las dos conectadas dejará de funcionar la del puerto USB. Por lo tanto, si posteriormente quieres cambiar el código desde el PC, tendrás que desconectar estos dos pines (o al menos el de RX).

4. Ejecuta el programa. El resultado ha de ser similar a:

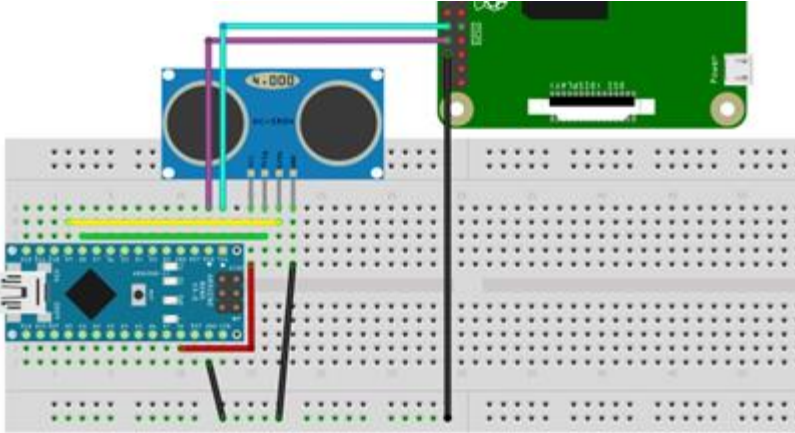
```

...: Lista de UART disponibles: [MINIUART, UART0]
...: Mandado a Arduino: H
...: 1 bytes escritos en UART
...: Recibido de Arduino: H

```

5. Conecta los pines del puerto UART entre la Raspberry Pi y Arduino, teniendo en cuenta que los bits TX y RX han de estar cruzados. Conecta también GND:

NOTA: Arduino Nano trabaja con 5V en las líneas de transmisión/recepción mientras que Raspberry Pi utiliza 3,3V. Si trabajas con estos dispositivos te recomendamos que leas el siguiente apartado. Otros microcontroladores como el ESP32 trabajan con 3,3V por lo que no sería necesaria esta conversión.



NOTA: cada procesador ha de alimentarse con su fuente de alimentación. No hay que interconectar las líneas de 5V o 3,3V. Solo se conectan las masas, de lo contrario no funcionaría el interfaz UART.

6. Ejecuta el programa. En el log ha de aparecer, la respuesta de Arduino frente a “H”, que ha de ser “Hola Mundo”:
7. Envía el comando “D” y comprueba que se recibe la distancia del sensor.



Preguntas de repaso: [Usar Microcontrolador \(Arduino\) como esclavo](#)

1.7.1. Aadar señales de 3,3V a 5V

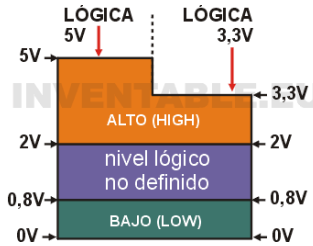
La forma clásica de trabajar consiste en utilizar 5V como nivel alto y 0V como nivel bajo. No obstante, se ha popularizado utilizar una lógica diferente donde el nivel alto está representado por 3,3V.

Como acabamos de ver es frecuente que tengamos que interconectar elementos que trabajan con una lógica y con o la otra. Existen convertidores de voltaje que podemos comprar para resolver este problema. También existen soluciones más sofisticadas⁸, pero en este apartado lo resolveremos de una forma muy sencilla. Hay que destacar que esta solución trabaja solo para líneas unidireccional, en la línea siempre transmite el mismo. Para una solución bidireccional, consultar el link.

Salida a 3,3V y entrada a 5V

Aunque se suele trabajar con un valor de voltaje como referencia a la hora de interpretar una entrada se utilizan un rango bastante amplio. Este rango se muestra en la siguiente gráfica:

⁸ <https://www.inventable.eu/2017/05/03/adaptadores-nivel-5v-3-3v/>

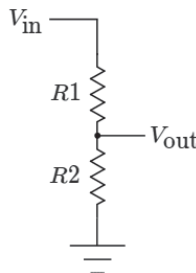


Como puedes verificar en caso de que la salida sea a 3,3V un circuito con lógica de 5V va a interpretarlo como nivel alto sin problemas. Tampoco hay problemas con la salida a nivel bajo.

Salida a 5V y entrada a 3,3V

Es en este caso cuando podemos tener problemas. Una salida a 5V podría dañar un circuito preparado para recibir un máximo de 3,3V. Algunos dispositivos que trabajan a 3,3V permiten entradas a 5V. En caso de duda podemos consultar las especificaciones técnicas en el datasheet.

Para resolver el problema queremos que un voltaje de 5V baje a 3,3V mientras que uno de 0V no cambie. Para ello podemos usar un simple divisor de tensión usando dos resistencias, tal y como se muestra en el siguiente esquema:



Donde $V_{in} = 5V$ y $V_{out} = 3,3V$. Para calcular $R1$ y $R2$ aplicando la ley de Ohm.

$$V = I \cdot R$$

Que nos dice que la caída de voltaje es proporcional a la Resistencia. Dado que V_{out} es 2/3 de V_{in} , $R2$ ha de ser el doble que $R1$. De esta forma en $R1$ caerá 1/3 del voltaje y en $R2$ los 2/3 restantes.

Existen infinitas resistencias que cumplen esta relación. Si estas tienen un valor muy pequeño, pasará mucha intensidad por el circuito, y si sus valores son grandes pasará poca. Un valor adecuado puede ser de 1mA, con lo que podemos calcular:

$$V = I \cdot R \Rightarrow R = V / I \Rightarrow$$

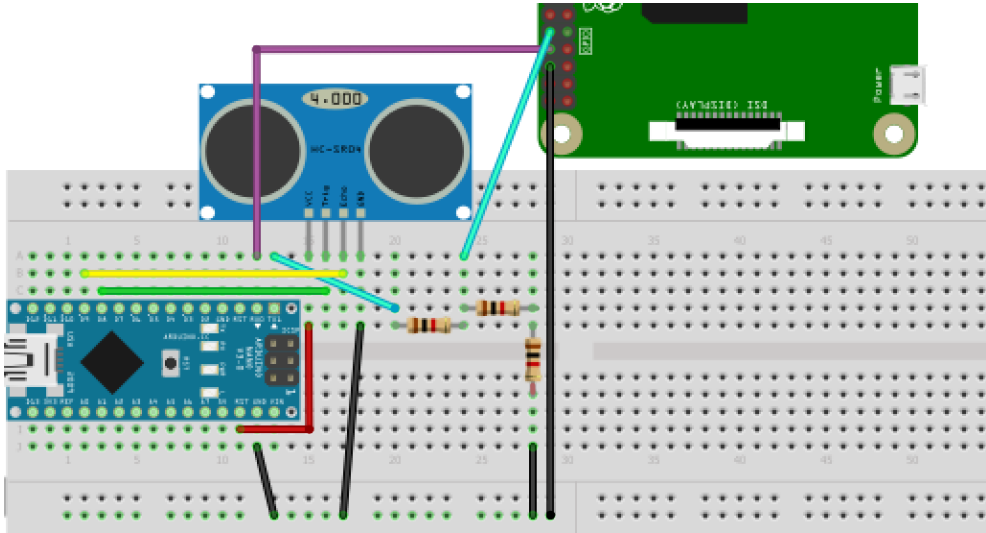
$$R2 = V_{out} / I = 3,3 / 0,001 = 3,3K$$

$$R1 = (V_{in} - V_{out}) / I = (5 - 3,3) / 0,001 = 1,7K$$

Seguramente no dispongas de resistencias con exactamente estos valores, pero no hay problema en que utilices unas similares. Por ejemplo, En el kit de la

RaspBerry Pi la más parecida es de 1K. En el siguiente esquema se ha utilizado una resistencia de 1K para R1 y dos de este valor en serie para R2 (equivale a 2K). Al usar resistencias más pequeñas la corriente del circuito será mayor. En concreto $I = V / R = 5V / 3K\Omega = 1,66mA$. Lo que no va a suponer ningún problema.

El circuito resultante será en siguiente:



1.8. Controladores de usuario

Como vimos en una sección anterior, interactuar con el chip PCF8591, puede ser un trabajo muy complejo. También vimos cómo utilizar un driver o controlador podría simplificar mucho este trabajo. No solo tenemos esta ventaja, además es muy posible que el driver realice las operaciones de una forma más correcta y está ampliamente testeado, de lo que podríamos hacer nosotros.

Android Things introduce un marco de desarrollo que nos facilita la utilización y creación de controladores, a través de `UserDriverManager`.

1.8.1. Utilizar controladores

Si quieres utilizar un controlador ya desarrollado sigue los siguientes pasos:

1. Busca el driver para tu dispositivo. Por ejemplo, en:
<https://github.com/androidthings/drivers-samples>
2. Añade la dependencia en el build.gradle:

```
dependencies {
    ...
    compile 'com.google.android.things.contrib:driver-button:1.0'
}
```

3. Pide el permiso para usar el tipo de driver en el manifiesto:

```
<uses-permission android:name="com.google.android.things.permission.MANAGE_INPUT_DRIVERS" />
```

Es muy posible que el driver requiera algún permiso adicional, como el de entradas y salidas.

4. Inicializa la clase del controlador con los parámetros adecuados.

```
import com.google.android.things.contrib.driver.button.Button;
import com.google.android.things.contrib.driver.button.ButtonInputDriver;
...
public class ButtonActivity extends Activity {
    private static final String NOMBRE_DE_PIN = ...;
    private ButtonInputDriver driverBoton;

    @Override protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        try {
            driverBoton = new ButtonInputDriver(NOMBRE_DE_PIN,
                Button.LogicState.PRESSED_WHEN_LOW, KeyEvent.KEYCODE_SPACE);
        } catch (IOException e) {
            Log.e(TAG, "Error configurando pin GPIO", e);
        }
    }
}
```

5. Recuerda cerrar el recurso cuando dejes de necesitarlo.

```
@Override protected void onDestroy(){
    super.onDestroy();
    if (driverBoton != null) {
        try {
            driverBoton.close();
        } catch (IOException e) {
            Log.e(TAG, "Error cerrando driver de botón ", e);
        }
    }
}
```

6. Activarlo y desactivarlo cuando se necesario:

```
@Override protected void onStart() {
    super.onStart();
    driverBoton.register();
}
@Override protected void onStop() {
    super.onStop();
    driverBoton.unregister();
}
```

7. El driver podrá interactuar con el sistema estándar Android. Por ejemplo, modificando la localización del dispositivo, o como en el ejemplo mostrado introducir teclas en la entrada estándar.

```
@Override public boolean onKeyDown(int keyCode, KeyEvent event) {
    if (keyCode == KeyEvent.KEYCODE_SPACE) {
        // ...
        return true;
    }
    return super.onKeyDown(keyCode, event);
}
```

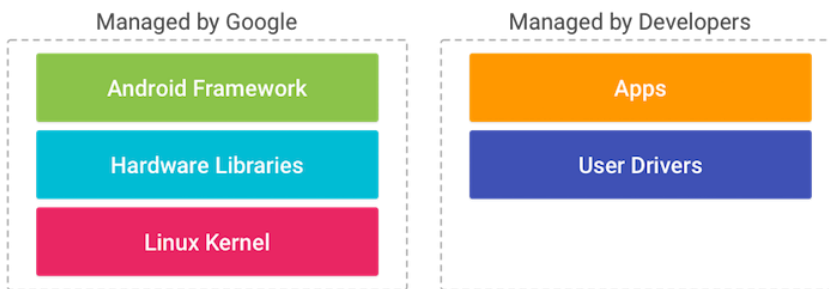
```

    }
    @Override public boolean onKeyUp(int keyCode, KeyEvent event) {
        if (keyCode == KeyEvent.KEYCODE_SPACE) {
            // ...
            return true;
        }
        return super.onKeyUp(keyCode, event);
    }
}
}

```

1.8.2. Escribir controladores de usuario

Para permitir a los desarrolladores de aplicaciones que puedan registrar nuevos controladores de dispositivos, Android Things introduce el concepto de un controlador de usuario (user driver). Los controladores de usuario son componentes registrados desde las aplicaciones que amplían los servicios existentes del framework Android. Permiten que cualquier aplicación inserte eventos de hardware en el framework que otras aplicaciones puedan procesar utilizando las API estándar de Android.



Permitir que las aplicaciones interactúen de forma directa con el framework puede hacer al sistema inestable e inseguro. También presenta otros beneficios muy necesarios en el marco de IoT, donde los tipos de dispositivos y periféricos son inmensos. Esta flexibilidad nos va a permitir que nuestro código sea portable, reutilizable y fácil de integrar.

Podemos diferenciar varios tipos de controladores:

Dispositivos de Interfaz Humana (HID): obtienen información proporcionada por el usuario. Algunos ejemplos son teclados, almohadillas táctiles y mando. Solicitar permiso: `MANAGE_INPUT_DRIVERS`. [Más info](#)

Ubicación: proporcionan información de ubicación física del dispositivo. Pueden basarse en diferentes tecnologías como GPS, WiFi, Bluetooth, ... Solicitar permiso: `MANAGE_GNSS_DRIVERS`. [Más info](#)

Sensores: miden las condiciones del entorno físico. Algunos controladores pueden fusionar información de varios sensores físicos en un solo sensor virtual. Esto es particularmente común con sensores de dirección, que combinan acelerómetros, giroscopios y campo magnético. Solicitar permiso: `MANAGE_SENSOR_DRIVERS`. [Más info](#)

LowPAN: las redes inalámbricas de área personal de baja potencia permiten que los dispositivos se conecten e intercambien datos a través de redes en malla. Los controladores LoWPAN te permiten integrar un módulo de radio externo al dispositivo para que siga las especificaciones de la API LoWPAN. Solicitar permiso: `MANAGE_LOWPAN_INTERFACE`. [Más info](#)

1.8.2.1. Escribir controladores de Interfaz de Usuario

Estos controladores van a tener el privilegio de insertar eventos de pulsación de teclas o eventos de movimiento sobre superficie táctil⁹. Requieren de la solicitud del permiso:

```
<uses-permission
    android:name="com.google.android.things.permission.MANAGE_INPUT_DRIVERS" />
```

Veamos un ejemplo, que permite introducir la tecla Escape:

1. Crea una instancia del driver utilizando `InputDriver.Builder`.
2. Registra el driver con `UserDriverManager`.

```
public class ButtonDriverService extends Service {
    private static final String DRIVER_NAME = "EscapeButton";
    private static final int KEY_CODE = KeyEvent.KEYCODE_ESCAPE;
    private InputDriver driver;

    @Override public void onCreate() {
        super.onCreate();
        driver = new InputDriver.Builder() // Creamos instancia de driver
            .setName(DRIVER_NAME)
            .setSupportedKeys(new int[] {KEY_CODE})
            .build();
        UserDriverManager manager = UserDriverManager.getInstance();
        manager.registerInputDriver(driver);
    }

    @Override public IBinder onBind(Intent intent) {
        return null;
    }
}
```

3. Cuando ocurra algún evento hardware por ejemplo se pulsa un botón, llama al siguiente método:

```
private void lanzarEvento(boolean pressed) {
    InputDriverEvent event = new InputDriverEvent();
    event.setKeyPressed(KEY_CODE, pressed);
    driver.emit(event);
}
```

4. Desactiva el registro del driver en `onDestroy()`:

⁹ <http://source.android.com/devices/input/overview.html>

```
@Override public void onDestroy() {
    super.onDestroy();
    DriverManager.getInstance().unregisterInputDriver(driver);
}
```

Puedes consultar la implementación del controlador `ButtonInputDriver` que hemos utilizado en la sección anterior en: <https://github.com/androidthings/contrib-drivers/tree/master/button>

Si en lugar de teclas quieres introducir eventos de movimiento, emulando que un usuario desliza un dedo sobre una pantalla táctil. En la creación del driver reemplaza las siguientes líneas:

```
driver = new InputDriver.Builder()
    .setName(DRIVER_NAME)
    .setAxisConfiguration(MotionEvent.AXIS_X, 0, 255, 0, 0)
    .setAxisConfiguration(MotionEvent.AXIS_Y, 0, 255, 0, 0)
    .build();
```

Y reemplaza el método:

```
private void lanzarEvento(int x, int y, boolean pressed) {
    InputDriverEvent event = new InputDriverEvent();
    event.setPosition(MotionEvent.AXIS_X, x);
    event.setPosition(MotionEvent.AXIS_Y, y);
    event.setContact(pressed);
    driver.emit(event);
}
```

1.8.2.2. Escribir controladores de Ubicación

Como veremos va a ser muy sencillo. Primero solicita el permiso:

```
<uses-permission
    android:name="com.google.android.things.permission.MANAGE_GNSS_DRIVERS" />
```

Utiliza como base el driver anterior. En la creación reemplaza:

```
driver = new GnssDriver();
```

Y utiliza el método:

```
private void lanzarEvento(DatosDePosicion datos) {
    Location location = obtenLocalización(datos);
    driver.reportLocation(location);
}
```



Preguntas de repaso: [Controladores de usuario en Android Things](#)

1.9. Integrar Google Assistant SDK

Google ha apostado muy fuerte por el control de dispositivos IoT por medio de la voz. Para facilitar la integración de esta tecnología en cualquier dispositivo ha creado *Google Assistant SDK for devices*.

Este SDK brinda dos opciones para integrar el Asistente en tu dispositivo:

Google Assistant Library

Esta biblioteca está escrita en Python y es compatible con dispositivos con arquitecturas linux-armv7l y linux-x86_64. La biblioteca expone una API de alto nivel basada en eventos que es fácil de ampliar. Proporciona las siguientes características lista para usar:

- Activación manos libres: con Hey Google o Ok Google , ¡al igual que con Google Home!
- Captura y reproducción de audio
- Gestión del estado de conversación
- Gestión de temporizador y alarma

Servicio Asistente de Google

Es la opción más flexible y con más amplio soporte. Expone una API de bajo nivel que manipula directamente los bytes de audio de una solicitud y respuesta de un asistente. Está basado en [gRPC](#)¹⁰, por lo que puede ser usado desde cualquier plataforma que los admita. No admite las características que acabamos de listar, por lo que tendrías que implementarlas por tu cuenta.

Acciones en Google

El SDK nos permitirle agregar una funcionalidad única para nuestro dispositivo con [Acciones en Google](#). Las acciones pueden ser registradas y creadas a través de la [consola de acciones](#).

Nombre	Dispositivos	Descripción	Frases
OnOff	cualquiera	conectar / desconectar el dispositivo	Turn on / Encender Turn off / Apagar
StartStop	cualquiera	arranque / parada general	Start the washing machine.

¹⁰ Sistema de llamadas a procedimiento remoto (RPC) de código abierto desarrollado por Google. Puede considerarse una alternativa a REST. Usa HTTP / 2.

Modes	cualquiera	dispositivos que pueden trabajar en diferentes modos.	What mode is the dryer in? Set the dryer to delicate.
Toggles	cualquiera	cambia botones físicos, asociados a una función.	Is my dryer sterilization on? Turn on sterilization for the dryer.
Brightness	Light	nivel de brillo de 0 a 100	Adjust my light to 65% brightness. / Ajusta mi luz a un 50% de brillo.
FanSpeed	Air conditioning unit , Air purifier , Fan	velocidad del ventilador	What speed are the fans in the living room?
TemperatureSetting	Air conditioning , Thermostat	temperatura de la habitación	Initialize Thermostat setting.
RunCycle	cualquiera	dispositivos vasados en ciclos de trabajo	What is the washing machine doing?
Locator	Vacuum , otros móviles	para preguntar ubicación del dispositivo.	Where are my keys?
...			

NOTA: En la actualidad estas acciones han sido definidas para los idiomas inglés (*en*), alemán (*de*), francés (*fr*) y japonés (*ja*). Incluso algunas en italiano (*it*). En la documentación se indica que todavía no se han traducido al español. No obstante, parece que desde junio de 2018 ya funcionan en castellano. En la tabla anterior se indican algunas traducciones que han sido verificadas.

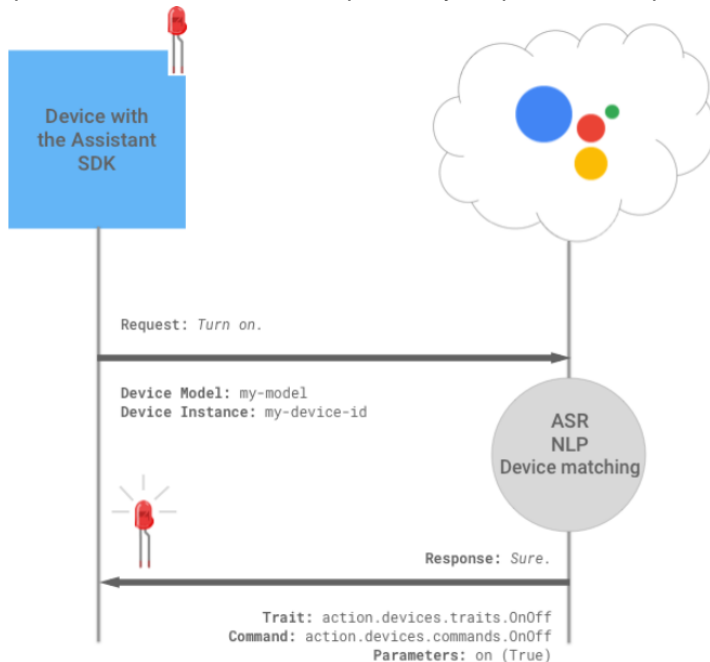
Para implementar un diálogo básico con el Asistente sigue estos pasos:

1. Implementa un cliente gRPC de transmisión de audio bidireccional.
2. Espera a que el usuario active una solicitud (por ejemplo, pulsando un botón).
3. Envía un mensaje [AssistRequest](#) de tipo `config` (ver [AssistConfig](#)) con los siguientes campos:
 - `audio_in_config`: formato del audio que se va a enviar. (ver [AudioInConfig](#))
 - `audio_out_config`: formato deseado para el audio recibido ([AudioOutConfig](#)).
 - `device_config`: dispositivo registrado en el Asistente (ver [DeviceConfig](#)).
 - `dialog_state_in`: estado del dialogo, ej. idioma, localización ([DialogStateIn](#)).
4. Empieza a grabar.
5. Envía mensajes [AssistRequest](#) con el audio de la consulta.
6. Recoge los mensajes [AssistResponse](#) entrantes.

7. Extrae los metadatos del mensaje. Por ejemplo, `conversation_state`, si se quiere cambiar el volumen o texto suplementario para visualizar en una pantalla (ver [DialogStateOut](#)).
8. Detén la grabación cuando reciba un [AssistResponse](#) con un `event_type` igual a `END_OF_UTTERANCE`.
9. Reproduce el audio de la respuesta del Asistente.
10. Toma el `conversation_state` que extrajiste antes y cópialo en el [DialogStateIn](#) del mensaje en siguiente [AssistRequest](#).

Si dispones de acciones específicas para tu dispositivo añade estos pasos:

1. En los mensajes entrantes, extrae el `device_action` (ver [DeviceAction](#)).
2. Analiza el campo JSON `device_request_json`. Consulta la página de [características](#) de cada tipo de dispositivo para ver la lista de características admitidas. Cada esquema de características muestra una solicitud EXECUTE de ejemplo con los comandos del dispositivo y los parámetros que se devuelven.



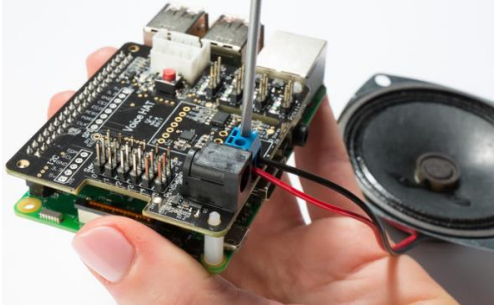
Si dispones de una entrada de texto, como un teclado, puedes enviar este texto en lugar de la voz en el campo `text_query` de [AssistConfig](#).

Para demostrarlo vamos a seguir el siguiente tutorial, aunque adaptando las entradas salida, y el idioma por defecto.

<https://codelabs.developers.google.com/codelabs/androidthings-assistant>

La mayor diferencia es que no vamos a necesitar en [Voice Kit](#). Compuesto por una Raspberry Pi Zero WH, altavoz, pulsador, caja de cartón y VoiceHat. Como se muestra en la siguiente figura, VoiceHat es una placa conectada al bus GPIO, para

entrada/salida de audio. Nosotros la reemplazaremos por una mini tarjeta de sonido por USB (imagen de la derecha). También podrías usar un micrófono por Bluetooth.

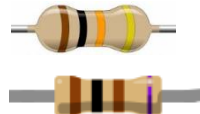
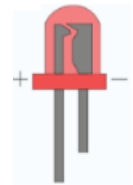


Ejercicio: *Preparar entradas y configuración de Google Assistant*

Corresponde a los pasos 2 y 3 del Codelab en que nos basamos.

Material necesario:

- Raspberry Pi 3B
- tarjeta SD con Android Things
- un pulsador
- una resistencia de pull-up¹¹ de 10 K Ω (color: café, negro, naranja).
- LED y resistencia de ajuste¹²
- tablero de prototipos y cables.
- Entrada salida audio por USB¹³
- Micrófono y auricular

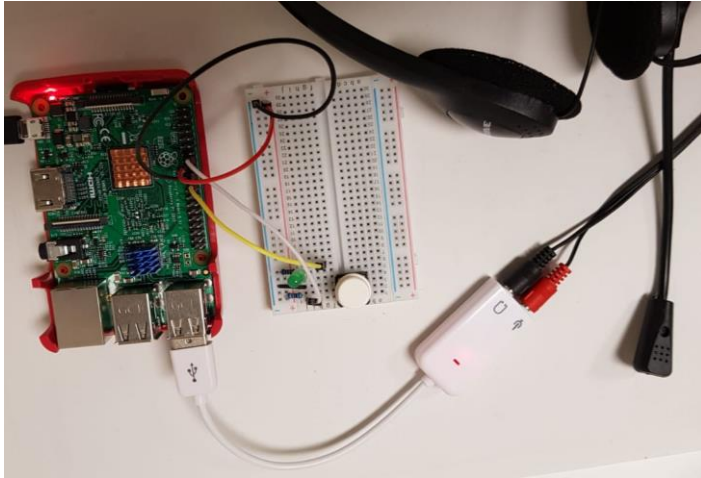


1. Conecta la tarjeta de sonido por USB a uno de los cuatro puertos que tiene la Raspberry Pi. Conecta a su vez el micrófono y auricular a la tarjeta.

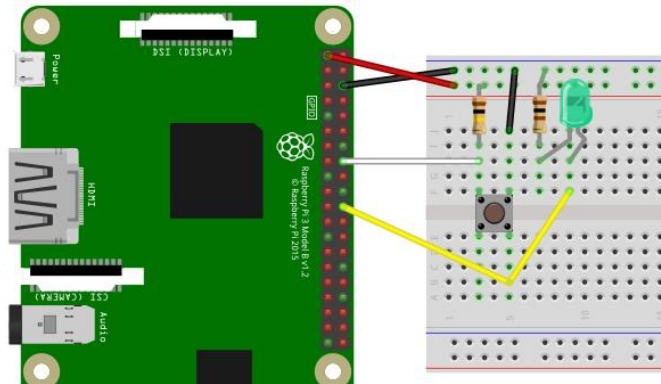
¹¹ Ver apartado: Qué es una resistencia pull-up

¹² Ver apartado: LED y resistencia de ajuste

¹³ Por ejemplo: <http://www.dx.com/p/bstuo-virtual-3d-stereo-7-1-channel-usb-sound-card-silver-472444#.WvRn0liFM2w>



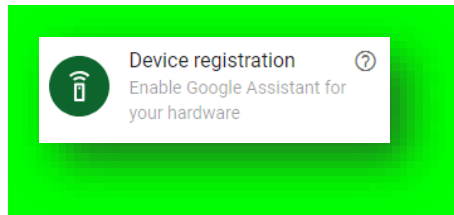
2. Monta el pulsador y el LED en la placa de prototipos y conectarlos a la Raspberry Pi. El siguiente esquema te muestra cómo hacerlo:



- Conecta una conexión del botón al pin de entrada GPIO BCM23. Conecta el mismo pin a 3.3V a través de una resistencia pull-up de 10 KΩ. Conecta el otro extremo del botón a tierra.
 - Conecta la salida GPIO BCM25 al ánodo del LED (pata más larga). Conecta el cátodo del LED (pata más corta) a tierra a través de una resistencia de ajuste (típicamente 100Ω para rojo, amarillo o verde y 10Ω para azul, violeta o blanco).
3. **Comenzamos dando algunos permisos en nuestra cuenta de Google necesarios para hacer el tutorial.** Entra en Controles de actividad de tu cuenta Google: <https://myaccount.google.com/activitycontrols>
 4. Activa las siguientes funciones:
 - Actividad en la Web y en Aplicaciones
 - Información del dispositivo
 - Actividad de Voz y Audio
 5. **Vamos a crear un nuevo proyecto en la consola de Acciones.** Abre la consola <https://console.actions.google.com>.

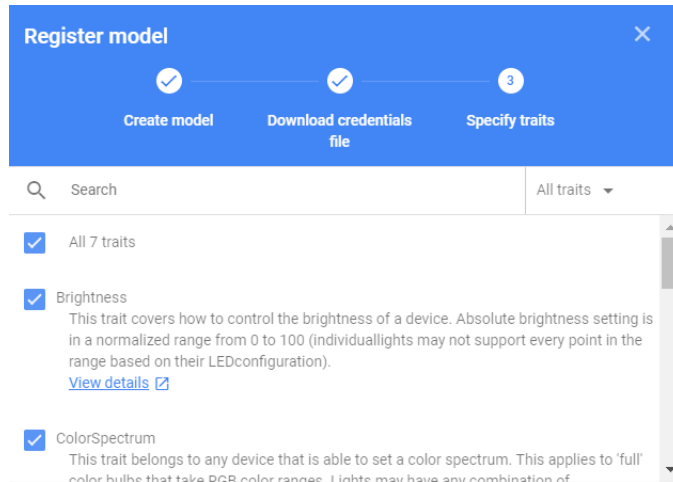
6. Crea un proyecto nuevo o selecciona uno existente.

7. Selecciona al final de la página la opción *Device registration*. Con esto vamos a habilitar el uso del asistente de voz para incorporarlo en un dispositivo hardware creado por nosotros.



8. Pulsa en **REGISTER MODEL** (puede tardar en activarse unos minutos) e introduce los siguientes valores:

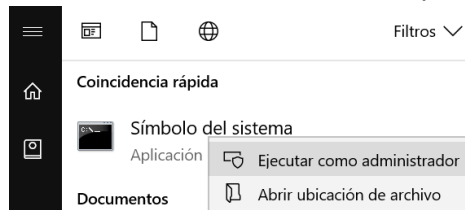
9. Pulsa en **REGISTER MODEL**. En el siguiente paso descarga el fichero con las credenciales y renómbralo como *credentials.json*, más tarde lo copiarás en la carpeta raíz de tu proyecto.
10. En el último paso selecciona las 7 características (traits) que puede tener un dispositivo tipo Ligh. No las vamos a implementar todas, pero al seleccionarl las vamos a poder interactuar con el asistente usando varias características.



11. Pulsa en **SAVE TRAITS**.
12. Selecciona en el menú de la izquierda la opción *Language*. Además de *English*, selecciona *Spanish*. De esta forma podremos dictar las frases en español.
13. Habilita [Google Assistant API](https://console.developers.google.com/apis/api/embeddedassistant.googleapis.com/overview) en la consola de Google APIs:
<https://console.developers.google.com/apis/api/embeddedassistant.googleapis.com/overview>
14. Entra en la pestaña *Pantalla de consentimiento de OAuth*, introduce el campo *Correo electrónico de asistencia* y pulsa en *Guardar*. ¿?
15. Desde esta consola puedes consultar las cuotas:

Nombre de cuota	Límite
(Unlabeled quota) por día	500
(Unlabeled quota) por minuto por usuario	60

16. Vamos a necesitar ejecutar algunos scripts en Python. Instala python3 desde <https://www.python.org/download/releases/3.0/>. Recuerda indicar que añada la aplicación al Path.
17. En el ordenador de desarrollo abre una consola con permiso de administrador.



18. Ejecuta el siguiente comando:

```
pip install --upgrade pip setuptools wheel
pip install --upgrade google-auth-oauthlib[tool]
```

De esta forma instalamos dos librerías necesarias.

19. Copia en la raíz del proyecto donde quieras añadir el asistente de Google el fichero *credentials.json*. *Nota: Si quieres seguir los tutoriales propuestos haz el punto 3 del siguiente ejercicio.*

20. **Abre un intérprete de comandos,** sitúate en la carpeta raíz del proyecto y ejecuta el siguiente comando:

```
google-oauthlib-tool --client-secrets credentials.json
--credentials shared/src/main/res/raw/credentials.json
--scope https://www.googleapis.com/auth/assistant-sdk-prototype --save
```

Esto abrirá un navegador y te pedirá que autorices que la aplicación solicite el asistente en tu nombre.

 Iniciar sesión con Google

project-1059772976484 quiere
acceder a tu cuenta de Google

 jtomas00@gmail.com

Esto permitirá a **project-1059772976484** hacer lo siguiente:

 Usar el Asistente de Google: amplio acceso a tu cuenta de Google ⓘ

Confirma que confías en **project-1059772976484**

Puede que estés compartiendo información sensible con este sitio web o esta aplicación. Consulta las condiciones del servicio y las políticas de privacidad de **project-1059772976484** para saber cómo se tratarán tus datos. Puedes ver o retirar el acceso en cualquier momento en tu [cuenta de Google](#).

[Más información sobre los riesgos](#)

Cancelar

Permitir

21. Verifica que en la carpeta `shared/src/main/res/raw` se ha creado un nuevo fichero `credentials.json`, aunque esto no es igual al descargado.



Ejercicio: Versión inicial de Google Assistant

Corresponde al paso 4 del Codelab y al módulo step1.

1. Arranca la Raspberry Pi y averigua su dirección IP.
2. Desde el ordenador de desarrollo ejecuta:

```
adb connect <ip-address>
```







3. Abre Android Studio y selecciona:

File/New/Project form Version Control/Git

Indica la siguiente URL:

<https://github.com/googlecodelabs/androidthings-googleassistant.git>

4. Verifica como el proyecto está dividido en cinco módulos:

- >  shared
- >  step1-start-here
- >  step2-volume-control
- >  step3-builtin-device-actions
- >  step4-custom-device-actions
- >  Gradle Scripts

Uno común (*shared*) y cuatro que corresponden a versiones cada vez más elaboradas del proyecto. Cada uno de los ejercicios propuestos utilizarán un módulo distinto.

5. En el módulo *shared* abre la clase *MyDevice*, introduce el valor correcto para *MODEL_ID* según lo obtenido en el ejercicio anterior.

```
public class MyDevice {
    public static final String MODEL_ID = "asistente-39d8f-producto-d91grk";
    public static final String INSTANCE_ID = "id_unica_por_dispositivo";
    public static final String LANGUAGE_CODE = "es-ES"; //"en-US"
}
```

En *INSTANCE_ID* has de introducir un valor diferente por cada dispositivo que instales. Indica también que quieres utilizar el idioma español.

6. La clase *BroadDefaults* permite definir los puertos GPIO del botón y del LED.
7. Los cuatro *AndroidManifest.xml* que encontrarás en cada módulo *stepX-...*, no instalan la actividad como la de inicio del dispositivo. Si quieres que al reiniciar la Raspberry Pi se ejecute automáticamente, añade las siguientes líneas:

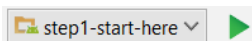
```
<intent-filter>
    <action android:name="android.intent.action.MAIN" />
    <category android:name="android.intent.category.IOT_LAUNCHER" />
    <category android:name="android.intent.category.DEFAULT" />
</intent-filter>
```

8. Otro cambio que has de realizar en *AssitantActivity* de los 4 módulos es cambiar la variable.

```
private static final boolean USE_VOICEHAT_DAC = true false;
```

Si no lo cambias, tratará de utilizar el utilizar el VoiceHat y no funcionará.

9. Ejecuta el módulo step1:



10. Mantén pulsado el botón mientras te comunicas con el asistente. Puedes preguntar la hora o cualquier otra información. Al soltar el botón escucharás la respuesta.
11. Si conectas un monitor podrás ver una lista de las frases reconocidas más probables.
12. Si estudias su código, verás cómo se activa el asistente:

```
@Override public void onButtonEvent(Button button, boolean pressed) {
    ...
    mLed.setValue(pressed);
    if (pressed) mAssistantHandler.post(mStartAssistantRequest);
}
```

```

    else                mAssistantHandler.post(mStopAssistantRequest);
}

```

13. El Runnable `mStartAssistantRequest` pone la grabación en marcha `mAudioRecord.startRecording()`. Luego configura todos los parámetros de asistente usando `converseConfigBuilder`.
14. Estudia el Runnable `mStreamAssistantRequest`.
15. Estudia el Runnable `mStopAssistantRequest`.

1.9.1. Añadir control de volumen

La API de Google Assistant permite controlar el volumen del dispositivo asistente a través de la voz con consultas como: "baja el volumen" o "pon el volumen a 6". Si prueba esas consultas con el proyecto de inicio, comprobarás que el Asistente todavía no las comprende. Hay que proporcionar información sobre el volumen actual del dispositivo antes de que el Asistente pueda actualizarlo.

En el siguiente ejercicio aprenderás a capturar estas acciones para poder configurar el volumen de la salida de audio del asistente.



Ejercicio: Añadir control de volumen

1. Abre el módulo step-2 del proyecto. Observa cómo se ha añadido la siguiente variable:

```
private static int mVolumePercentage = 100;
```

2. A la hora de configurar el audio de salida haz el siguiente cambio: (a diferencia del resto de paso, que ya están hechos, este no lo está)

```

AssistConfig.Builder converseConfigBuilder = AssistConfig.newBuilder()
    .setAudioInConfig(ASSISTANT_AUDIO_REQUEST_CONFIG)
    .setAudioOutConfig(ASSISTANT_AUDIO_RESPONSE_CONFIG)
    .setAudioOutConfig(AudioOutConfig.newBuilder()
        .setEncoding(ENCODING_OUTPUT)
        .setSampleRateHertz(SAMPLE_RATE)
        .setVolumePercentage(mVolumePercentage)
        .build())
    .setDeviceConfig(DeviceConfig.newBuilder()
        .setDeviceModelId(MyDevice.MODEL_ID)
        .setDeviceId(MyDevice.INSTANCE_ID)
        .build());

```

3. En el método `onNext()` se procesan las acciones reconocidas por el asistente. En caso de ser necesario actualizaremos el volumen utilizando `mAudioTrack.setVolume()`:

```

@Override public void onNext(AssistResponse value) {
    ...
    if (value.getDialogStateOut() != null) {
        int volume = value.getDialogStateOut().getVolumePercentage();
        if (volume > 0) {
            mVolumePercentage = volume;
        }
    }
}

```

```

        Log.i(TAG, "assistant volume changed: " + mVolumePercentage);
        mAudioTrack.setVolume(AudioTrack.getMaxVolume() *
                               mVolumePercentage / 100.0f);
    }
    mConversationState= vaue.getDialogStateOut().getConversationState();
}
...

```

4. Ejecuta el módulo step-2. Dicta la frase “pon el volumen a 2”.
5. Verifica que en el logCat aparece: `assistant volume changed: 20`

NOTA: Aunque el comando es reconocido es posible que no aprecies un cambio en el volumen. Puede ser algún problema de compatibilidad con la tarjeta de sonido.

NOTA: El volumen se restablecerá cada vez que reinicie el dispositivo. Utiliza `SharedPreferences` para guardar este nuevo volumen.

1.9.2. Usar acciones predefinidas en Google Assistant

Cuando utilizamos el Asistente de Google para interactuar con nuestro dispositivo, es posible que queramos realizar ciertas acciones, como encenderlo, apagarlo, configurar su brillo, ... Para hacer esto, puede usar acciones predefinidas. Podemos configurar el asistente para que nuestro dispositivo reciba alguna de estas acciones, y así podamos actuar en consecuencia.

Este esquema de interacción se conoce como Acciones (Actions). Puedes encontrar una descripción más detallada en:

<https://developers.google.com/actions/smarthome/>

Cuando diga un comando que su dispositivo puede admitir, recibirá una carga JSON que le permitirá manejar la consulta directamente.

Cada dispositivo va a tener una serie de características (device traits) que podremos modificar con cada acción. Al principio de este apartado se muestra una tabla con algunas de estas características. La tabla completa puede consultarse en <https://developers.google.com/actions/smarthome/traits/>.

Como veremos en el siguiente ejercicio, el primer paso va a ser dar de alta nuestro dispositivo concreto. Así Google Assistant sabrá si este dispositivo tiene la luz encendida o apagada:



Ejercicio: Usar acciones predefinidas en Google Assistant

1. En el ejercicio anterior configuramos ciertas características para nuestro dispositivo. Aunque en el tutorial original, te piden que lo hagas ahora, nosotros hemos decidido hacerlo antes.

NOTA: Si modificas las características sobre un modelo ya creado es posible que estas no se activen. Si tienes este problema, crea un nuevo modelo y borra el anterior.

2. En el ordenador de desarrollo abre una consola con permiso de administrador. Sitúate en la carpeta raíz del proyecto y ejecuta el siguiente comando:

```
google-oauthlib-tool --client-secrets credentials.json --scope
https://www.googleapis.com/auth/assistant-sdk-prototype --save
```

Se abrirá una página Web y te pedirá permiso correspondiente.

3. Ejecuta el siguiente comando, para instalar la herramienta:

```
pip install google-assistant-sdk
```

4. Ejecuta el siguiente comando, reemplazando tu project-id:

```
googlesamples-assistant-devicetool --project-id asistente-39d8f list --
model
```

Se mostrará una información similar a:

```
Device Model ID: asistente-39d8f-nuevo-d91grk
Project ID: asistente-39d8f
Device Type: action.devices.types.LIGHT
Trait action.devices.traits.Brightness
Trait action.devices.traits.ColorSpectrum
Trait action.devices.traits.ColorTemperature
Trait action.devices.traits.Dock
Trait action.devices.traits.OnOff
Trait action.devices.traits.StartStop
Trait action.devices.traits.TemperatureSetting
```

5. Ejecuta el siguiente comando, reemplazando los ids marcados:

```
googlesamples-assistant-devicetool --project-id asistente-39d8f register-
device --model asistente-39d8f-nuevo-d91grk --device ins-
tance_id_unica_unica_por_dispositivo_2 --client-type SERVICE
```

6. Te indicará el mensaje “Device instance ... registered”
7. Cuando un usuario pronuncie alguna frase que coincida con alguna de las acciones que modifique alguna característica activada, nuestra aplicación será notificada. En concreto se enviará un JSON similar al siguiente:

```
{ "requestId": "ff36a3cc-ec34-11e6-b1a0-64510650abcf",
  "inputs": [{
    "intent": "action.devices.EXECUTE",
    "payload": {
      "commands": [{
        "devices": [{ "id": "123" }],
        "execution": [{
          "command": "action.devices.commands.OnOff",
          "params": { "on": true }
        }]
      }]
    }
  ]
}
```

8. Para tratar esta respuesta se utiliza el siguiente código, que encontrarás en el módulo step-3 del proyecto.:

```
@Override public void onNext(AssistResponse value) {
    if (value.getDeviceAction() != null &&
```

```

        !value.getDeviceAction().getDeviceRequestJson().isEmpty()) {
    try {
        JSONObject deviceAction = new JSONObject(value.getDeviceAction()
            .getDeviceRequestJson());
        JSONArray inputs = deviceAction.getJSONArray("inputs");
        for (int i = 0; i < inputs.length(); i++) {
            if (inputs.getJSONObject(i).getString("intent")
                .equals("action.devices.EXECUTE")) {
                JSONArray commands = inputs.getJSONObject(i)
                    .getJSONObject("payload")
                    .getJSONArray("commands");
                for (int j = 0; j < commands.length(); j++) {
                    JSONArray execution = commands.getJSONObject(j)
                        .getJSONArray("execution");
                    for (int k = 0; k < execution.length(); k++) {
                        String command = execution.getJSONObject(k)
                            .getString("command");
                        JSONObject params = execution.getJSONObject(k)
                            .optJSONObject("params");
                        handleDeviceAction(command, params);
                    }
                }
            }
        }
    } catch (JSONException | IOException e) {
        e.printStackTrace();
    }
}

```

9. Para tratar cada uno de los comandos, se utiliza el siguiente método:

```

public void handleDeviceAction(String command, JSONObject params)
    throws JSONException, IOException {
    // El parametro que envia ha cambiado desde lo que pone en el ejemplo
    //if (command.equals("action.devices.traits.OnOff")) {
    if (command.equals("action.devices.commands.OnOff")) {
        mLed.setValue(params.getBoolean("on"));
    }
}

```

Si el comando detectado es de tipo OnOff, extraemos el parámetro que indica el estado y encendemos o apagamos el LED según este parámetro.

NOTA: Si en el código aparece "action.devices.traits.OnOff" reemplaza "traits" por "commands".

10. Ejecuta el módulo step-3 del proyecto. Pulsa el botón y di "Encender"/ "Turn on". Te contestará "vale" y el LED se encenderá un instante. Pulsa el botón y di "Apagar"/ "Turn off"; Te contestará "vale". Al estar el LED ya apagado no se apreciará nada.



Práctica: *Control de un segundo LED por voz*

En el tutorial anterior se usa el mismo LED para visualizar el estado del botón y para realizar la acción. Como acabamos de comprobar no resulta nada claro. Para resolver este inconveniente utiliza un segundo LED para controlar su encendido y deja el LED actual para verificar la pulsación del botón.



Práctica: *Control del brillo del LED por voz*

Conecta el segundo LED a una salida PWM. Cuando el usuario diga la frase "Ajusta mi luz a un 60% de brillo" / "Adjust my light to 60% brightness" el LED ha de estar en estado alto el 60% del tiempo.

1.9.3. Definir acciones personalizadas

Es posible que necesites que tu dispositivo realice ciertas acciones no recogidas en las características que acabamos de exponer. Para hacer esto, puede usar [acciones de dispositivos personalizados](#).



Vídeo[Tutorial]: *Introduction to Custom Actions for the Google Assistant SDK*

Estas acciones se pueden definir con una gramática personalizada que puede incluir parámetros. Cuando diga un comando que su dispositivo puede admitir, recibirá una respuesta JSON que te permitirá manejar la consulta.

El siguiente ejemplo define una acción personalizada para conseguir un parpadeo en el LED. Podemos especificar que parpadee un número determinado de veces, así como la frecuencia que se definirá como un tipo personalizado.

```
{
  "manifest": {
    "displayName": "Blinky light",
    "invocationName": "Blinky light",
    "category": "PRODUCTIVITY" },
    "locale": "es",
    "actions": [ {
      "name": "com.example.actions.BlinkLight",
      "availability": {
        "deviceClasses": [ { "assistantSdkDevice": {} } ] },
      "intent": {
        "name": "com.example.intents.BlinkLight",
        "parameters": [
          { "name": "number", "type": "SchemaOrg_Number" } ],
      }
    } ]
  }
```

```

    { "name": "speed", "type": "Speed" } ],
    "trigger": {
      "queryPatterns": [
        "parpadea ($Speed:speed)? $SchemaOrg_Number:number veces",
        "parpadea $SchemaOrg_Number:number veces ($Speed:speed)?" ] }
  },
  "fulfillment": {
    "staticFulfillment": {
      "templatedResponse": {
        "items": [
          { "simpleResponse": {
              "textToSpeech": "Parpadeando $speed.raw $number veces"
            } },
          { "deviceExecution": {
              "command": "com.example.commands.BlinkLight",
              "params": { "speed": "$speed",
                "number": "$number" }}}
        ]
      }
    }
  }
},
"types": [
  { "name": "$Speed",
    "entities": [
      { "key": "slowly",
        "synonyms": [ "lentamente", "lento", "despacio" ] },
      { "key": "normally",
        "synonyms": [ "normal", "regular" ] },
      { "key": "quickly",
        "synonyms": [ "rápidamente", "rápido" ] }
    ]
  }
]
}

```



Ejercicio: Definir acciones personalizadas en Google Assistant

1. Copia el código JSON anterior en el fichero `actions.es.json`. Guárdalo en la raíz del proyecto. Asegúrate que se almacena en codificación UTF-8.

NOTA: Si te funciona el ejercicio, pero no reconoce ninguna palabra con acento o ñ, seguramente tendrás que cambiar la codificación del fichero.

2. Descarga la herramienta `gactions`¹⁴. Puedes dejar el fichero ejecutable (`gactions.exe`) en la misma carpeta del proyecto.

¹⁴ <https://developers.google.com/actions/tools/gactions-cli>

3. Con esta herramienta ejecuta (reemplazando el id de proyecto):

```
gactions update --action_package actions.es.json --project asistente-39d8f
gactions test --action_package actions.es.json --project asistente-39d8f
```

Te dará una URL de autorización donde podrás obtener un código de validación. Con esto tu dispositivo podrá recibir el comando definido en "com.example.commands.BlinkLight".¹⁵

4. Dentro de módulo step-4 se ha añadido el siguiente código. Corrige el error que se muestra:

```
public void handleDeviceAction(String command, JSONObject params)
    throws JSONException, IOException {
    mLedHandler.removeCallbacksAndMessages(null);
    if (command.equals("action.devices.traits.commands.OnOff")) { //ERROR
        mLed.setValue(params.getBoolean("on"));
    } else if (command.equals("com.example.commands.BlinkLight")) {
        int delay = 1000;
        int blinkCount = params.getInt("number");
        String speed = params.getString("speed");
        if (speed.equals("slowly")) delay = 2000;
        else if (speed.equals("quickly")) delay = 500;
        for (int i = 0; i < blinkCount * 2; i++) {
            mLedHandler.postDelayed(() -> {
                try {
                    mLed.setValue(!mLed.getValue());
                } catch (IOException e) {
                    throw new RuntimeException(e);
                }
            }, i * delay);
        }
    }
}
```

5. Ejecuta el módulo step-4 del proyecto. Pulsa el botón y di "parpadea 4 veces". El LED ha de encenderse y apagarse 4 veces.
6. Pulsa el botón y di "parpadea rápido 4 veces". Ha de hacer lo mismo, pero de forma más rápida.

NOTA: En caso de problemas te recomendamos que mires el siguiente tutorial <https://developers.google.com/assistant/sdk/guides/service/python/extend/custom-actions>

¹⁵ Si más adelante cambias el fichero *actions.json* has de borrar el fichero *creds.data* que se habrá creado, y ejecutar de nuevo estos comandos. CREO QUE NO HACE FALTA



Práctica: Acciones personalizadas en inglés y castellano

En esta práctica trataremos de funcione simultáneamente las acciones personalizadas en inglés y castellano.

1. Crea el fichero `actions.en.json` con las acciones en inglés, usando el código propuesto en <https://codelabs.developers.google.com/codelabs/androidthings-assistant/#6>
2. Añade en este fichero el atributo `locale`:

```
{ "manifest": {...},
  "locale": "en",
  "actions": [ {...}
]
```

3. Para eliminar las acciones anteriores borra el fichero `creds.data` que se habrá creado en la raíz del proyecto. (NO HACE FALTA)
4. Ejecuta los siguientes comandos (reemplazando el id de proyecto):

```
gactions update --action_package actions.en.json --action_package
actions.es.json --project asistente-39d8f (NO HACE FALTA)
gactions test --action_package actions.en.json --action_package
actions.es.json --project asistente-39d8f
```

5. Ejecuta el módulo `step-4` del proyecto. Pulsa el botón y di “parpadea 4 veces”. El LED ha de encenderse y apagarse 4 veces.
6. Pulsa el botón y di “blink 4 times”. Ha de hacer lo mismo.

NO ME FUNCIONA SIMULTANEAMENTE: Si pongo

```
public static final String LANGUAGE_CODE = "es-ES";
```

va en castellano

Si pongo

```
public static final String LANGUAGE_CODE = "en-EN";
```

va en inglés



Preguntas de repaso: [Google Assistant SDK en Android Things](#)