Bluetooth

La plataforma de Android incluye compatibilidad con la pila de red Bluetooth, la cual permite que un dispositivo intercambie datos de manera inalámbrica con otros dispositivos Bluetooth. El framework de la aplicación proporciona acceso a la funcionalidad Bluetooth mediante las Android Bluetooth API. Estas API permiten a las aplicaciones conectarse de manera inalámbrica con otros dispositivos Bluetooth y habilitan las funciones inalámbricas punto a punto y de multipunto.

Con las Bluetooth API, una aplicación de Android puede realizar lo siguiente:

- buscar otros dispositivos Bluetooth;
- consultar el adaptador local de Bluetooth en busca de dispositivos Bluetooth sincronizados;
- establecer canales RFCOMM;
- conectarse con otros dispositivos mediante el descubrimiento de servicios;
- transferir datos hacia otros dispositivos y desde estos;
- administrar varias conexiones.

En este documento, se describe la manera de usar el sistema *Bluetooth clásico*. Este sistema es la opción ideal para las operaciones que consumen más batería, como la transmisión y la comunicación entre dispositivos Android. Para dispositivos Bluetooth con bajos requisitos de energía, Android 4.3 (nivel 18 de API) presenta compatibilidad de API con Bluetooth de bajo consumo. Para obtener más información, consulta Bluetooth de bajo consumo (https://developer.android.com/guide/topics/connectivity/bluetooth-le.html).

Conceptos básicos

En este documento, se describe la manera de usar las Android Bluetooth API a fin de llevar a cabo las cuatro tareas principales necesarias para establecer la comunicación a través de Bluetooth: configuración de Bluetooth, búsqueda de dispositivos sincronizados o disponibles en el área local, conexión con dispositivos y transferencia de datos entre dispositivos.

Todas las Bluetooth API están disponibles en el paquete de `android.bluetooth` (https://developer.android.com/reference/android/bluetooth/package-summary.html). A continuación, se ofrece un resumen de las clases e interfaces que necesitarás para crear conexiones Bluetooth:

`BluetoothAdapter` (https://developer.android.com/reference/android/bluetooth/BluetoothAdapter.html)

> Representa el adaptador local de Bluetooth (radio Bluetooth). El `BluetoothAdapter` (https://developer.android.com/reference/android/bluetooth/BluetoothAdapter.html) es el punto de entrada de toda interacción de Bluetooth. Gracias a esto, puedes ver otros dispositivos Bluetooth, consultar una lista de los dispositivos conectados (sincronizados), crear una instancia de `BluetoothDevice` (https://developer.android.com/reference/android/bluetooth/BluetoothDevice.html) mediante una dirección MAC conocida y crear un `BluetoothServerSocket`

En este documento:

Clases fundamentales

- `BluetoothAdapter`
- `BluetoothDevice`
- `BluetoothSocket`
- `BluetoothServerSocket`

Ejemplos relacionados

- Chat de Bluetooth
- HDP (perfil de dispositivos de salud) Bluetooth

`BluetoothDevice` (https://developer.android.com/reference/android/bluetooth/BluetoothDevice.html)

Representa un dispositivo Bluetooth remoto. Usa esto para solicitar una conexión con un dispositivo remoto mediante un `BluetoothSocket` (https://developer.android.com/reference/android/bluetooth/BluetoothSocket.html) o consultar información sobre el dispositivo, como su nombre, dirección, clase y estado de conexión.

`BluetoothSocket` (https://developer.android.com/reference/android/bluetooth/BluetoothSocket.html)

Representa la interfaz de un socket de Bluetooth (similar a un `Socket` (https://developer.android.com/reference/java/net/Socket.html) de TCP). Este es el punto de conexión que permite que una aplicación intercambie datos con otro dispositivo Bluetooth a través de InputStream y OutputStream.

`BluetoothServerSocket` (https://developer.android.com/reference/android/bluetooth/BluetoothServerSocket.html)

Representa un socket de servidor abierto que recibe solicitudes entrantes (similar a un `ServerSocket` (https://developer.android.com/reference/java/net/ServerSocket.html) de TCP). A fin de conectar dos dispositivos Android, un dispositivo debe abrir un socket de servidor con esta clase. Cuando un dispositivo Bluetooth remoto realice una solicitud de conexión a este dispositivo, el `BluetoothServerSocket` (https://developer.android.com/reference/android/bluetooth/BluetoothServerSocket.html) mostrará un `BluetoothSocket` (https://developer.android.com/reference/android/bluetooth/BluetoothSocket.html) conectado una vez que la conexión se acepte.

`BluetoothClass` (https://developer.android.com/reference/android/bluetooth/BluetoothClass.html)

Describe las características y capacidades generales de un dispositivo Bluetooth. Se trata de un conjunto de propiedades de solo lectura que define las clases del dispositivo mayor y menor y sus servicios. Sin embargo, esto no describe de manera confiable todos los perfiles de Bluetooth y los servicios compatibles con el dispositivo, pero resulta útil como sugerencia para el tipo de dispositivo.

`BluetoothProfile` (https://developer.android.com/reference/android/bluetooth/BluetoothProfile.html)

Interfaz que representa un perfil de Bluetooth. Un *perfil de Bluetooth* es una especificación de interfaz inalámbrica para la comunicación entre dispositivos basada en Bluetooth. Un ejemplo es el perfil de manos libres. Para obtener más información sobre los perfiles, consulta Trabajo con perfiles (#Profiles).

`BluetoothHeadset` (https://developer.android.com/reference/android/bluetooth/BluetoothHeadset.html)

Brinda compatibilidad para que el uso de auriculares Bluetooth con teléfonos móviles. Esto incluye los perfiles de manos libres (v1.5) y de auriculares Bluetooth.

`BluetoothA2dp` (https://developer.android.com/reference/android/bluetooth/BluetoothA2dp.html)

Define la manera en que puede transmitirse audio de alta calidad de un dispositivo a otro a través de la conexión Bluetooth. “A2DP” significa “perfil de distribución de audio avanzada”.

`BluetoothHealth` (https://developer.android.com/reference/android/bluetooth/BluetoothHealth.html)

`BluetoothHealthCallback`
(https://developer.android.com/reference/android/bluetooth/BluetoothHealthCallback.html)

> Clase abstracta que se usa para implementar callbacks de `BluetoothHealth` (https://developer.android.com/reference/android/bluetooth/BluetoothHealth.html). Debes extender esta clase e implementar los métodos de callback para recibir actualizaciones sobre los cambios en el estado de registro de la aplicación y el estado de canal de Bluetooth.

`BluetoothHealthAppConfiguration`
(https://developer.android.com/reference/android/bluetooth/BluetoothHealthAppConfiguration.html)

> Representa una configuración de aplicación que la aplicación de salud de terceros de Bluetooth registra para comunicarse con un dispositivo de salud Bluetooth remoto.

`BluetoothProfile.ServiceListener`
(https://developer.android.com/reference/android/bluetooth/BluetoothProfile.ServiceListener.html)

> Interfaz que notifica a los clientes de IPC del `BluetoothProfile` (https://developer.android.com/reference/android/bluetooth/BluetoothProfile.html) cuando se conectan al servicio o se desconectan de este (es decir, el servicio interno que ejecuta un perfil en particular).

Permisos de Bluetooth

A fin de usar las funciones de Bluetooth en tu aplicación, debes declarar el permiso de Bluetooth `BLUETOOTH` (https://developer.android.com/reference/android/Manifest.permission.html#BLUETOOTH). Necesitas este permiso para establecer cualquier comunicación de Bluetooth, como solicitar o aceptar una conexión y transferir datos.

Si deseas que tu app inicie la detección de dispositivos o controle los ajustes de Bluetooth, también debes declarar el permiso `BLUETOOTH_ADMIN` (https://developer.android.com/reference/android/Manifest.permission.html#BLUETOOTH_ADMIN) . La mayoría de las aplicaciones necesitan este permiso solamente para poder ver dispositivos Bluetooth locales. Las demás funciones que otorga este permiso no deben usarse, a menos que la aplicación sea un “administrador de energía” que modifique los ajustes de Bluetooth a pedido del usuario. **Nota:** si usas el permiso `BLUETOOTH_ADMIN` (https://developer.android.com/reference/android/Manifest.permission.html#BLUETOOTH_ADMIN), también debes contar con el permiso `BLUETOOTH` (https://developer.android.com/reference/android/Manifest.permission.html#BLUETOOTH).

Declara los permisos de Bluetooth del archivo de manifiesto de tu aplicación. Por ejemplo:

```
<manifest ... >
  <uses-permission android:name="android.permission.BLUETOOTH" />
  ...
</manifest>
```

Consulta la referencia <uses-permission> (https://developer.android.com/guide/topics/manifest/uses-permission-element.html) para obtener más información sobre cómo declarar permisos de la aplicación.

Configuración de Bluetooth

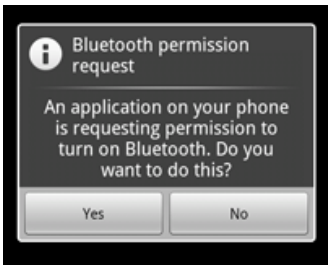

Figura 1: Habilitación del diálogo de Bluetooth.

Para que tu aplicación pueda comunicarse a través de Bluetooth, debes verificar que Bluetooth sea compatible con el dispositivo y, si es así, asegurarte de que esté habilitado.

Si Bluetooth no es compatible, debes inhabilitar correctamente cualquier función de Bluetooth. En caso de que Bluetooth sea compatible, pero esté inhabilitado, puedes solicitar que el usuario lo habilite sin abandonar tu aplicación. Esta configuración se logra en dos pasos, mediante el `BluetoothAdapter` (https://developer.android.com/reference/android/bluetooth/BluetoothAdapter.html).

1. Obtén el `BluetoothAdapter` (https://developer.android.com/reference/android/bluetooth/BluetoothAdapter.html)

   El `BluetoothAdapter` (https://developer.android.com/reference/android/bluetooth/BluetoothAdapter.html) es obligatorio para toda actividad de Bluetooth. Para obtener el `BluetoothAdapter` (https://developer.android.com/reference/android/bluetooth/BluetoothAdapter.html), llama al método estático del `getDefaultAdapter()` (https://developer.android.com/reference/android/bluetooth/BluetoothAdapter.html#getDefaultAdapter()). Esto muestra un `BluetoothAdapter` (https://developer.android.com/reference/android/bluetooth/BluetoothAdapter.html) que representa el propio adaptador de Bluetooth del dispositivo (la radio Bluetooth). Existe un adaptador de Bluetooth para todo el sistema y tu aplicación puede interactuar con él usando este objeto. Si `getDefaultAdapter()` (https://developer.android.com/reference/android/bluetooth/BluetoothAdapter.html#getDefaultAdapter()) muestra null, significa que el dispositivo no es compatible con Bluetooth y que tu tarea finaliza aquí. Por ejemplo:

   ```
   BluetoothAdapter mBluetoothAdapter = BluetoothAdapter.getDefaultAdapter();
   if (mBluetoothAdapter == null) {
       // Device does not support Bluetooth
   }
   ```

2. Habilita Bluetooth

   A continuación, debes asegurarte de que Bluetooth esté habilitado. Llama al `isEnabled()` (https://developer.android.com/reference/android/bluetooth/BluetoothAdapter.html#isEnabled()) para verificar si Bluetooth se encuentra actualmente habilitado. Si este método muestra false, Bluetooth no estará habilitado. Para solicitar la habilitación de Bluetooth, llama a `startActivityForResult()` (https://developer.android.com/reference/android/app/Activity.html#startActivityForResult(android.content.Intent, int)) con la intent de acción `ACTION_REQUEST_ENABLE` (https://developer.android.com/reference/android/bluetooth/BluetoothAdapter.html#ACTION_REQUEST_ENABLE). Esto generará una solicitud para habilitar Bluetooth a través de los ajustes de sistema (sin detener tu aplicación). Por ejemplo:

   ```
   if (!mBluetoothAdapter.isEnabled()) {
       Intent enableBtIntent = new Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
       startActivityForResult(enableBtIntent, REQUEST_ENABLE_BT);
   }
   ```

   Se mostrará un diálogo, como en la figura 1, en el que solicitará permiso al usuario para habilitar Bluetooth. Si el usuario responde “Sí”, el sistema comenzará a habilitar Bluetooth y el enfoque volverá a tu aplicación una vez que el proceso se complete con éxito (o no).

   La constante `REQUEST_ENABLE_BT` pasada a `startActivityForResult()` (https://developer.android.com/reference/android/app/Activity.html#startActivityForResult(android.content.Intent, int)) es un valor entero definido localmente (debe ser superior a 0) que el sistema te devuelve en tu implementación de `onActivityResult()`

Si la habilitación de Bluetooth se realiza con éxito, tu actividad recibe el código de resultado `RESULT_OK` (https://developer.android.com/reference/android/app/Activity.html#RESULT_OK) en el callback `onActivityResult()` (https://developer.android.com/reference/android/app/Activity.html#onActivityResult(int, int, android.content.Intent)). En caso de que Bluetooth no se habilitara debido a un error (o a que el usuario respondiera "No"), el código de resultado será `RESULT_CANCELED` (https://developer.android.com/reference/android/app/Activity.html#RESULT_CANCELED).

Opcionalmente, tu aplicación también puede recibir la intent de transmisión `ACTION_STATE_CHANGED` (https://developer.android.com/reference/android/bluetooth/BluetoothAdapter.html#ACTION_STATE_CHANGED), que el sistema transmitirá cada vez que se modifique el estado de Bluetooth. Esta transmisión contiene los campos extra`EXTRA_STATE` (https://developer.android.com/reference/android/bluetooth/BluetoothAdapter.html#EXTRA_STATE) y `EXTRA_PREVIOUS_STATE` (https://developer.android.com/reference/android/bluetooth/BluetoothAdapter.html#EXTRA_PREVIOUS_STATE), que incluyen los estados de Bluetooth nuevo y antiguo, respectivamente. Los valores posibles para estos campos extra son `STATE_TURNING_ON` (https://developer.android.com/reference/android/bluetooth/BluetoothAdapter.html#STATE_TURNING_ON), `STATE_ON` (https://developer.android.com/reference/android/bluetooth/BluetoothAdapter.html#STATE_ON), `STATE_TURNING_OFF` (https://developer.android.com/reference/android/bluetooth/BluetoothAdapter.html#STATE_TURNING_OFF) y `STATE_OFF` (https://developer.android.com/reference/android/bluetooth/BluetoothAdapter.html#STATE_OFF). La recepción de esta transmisión puede ser útil para detectar cambios realizados en el estado de Bluetooth mientras tu app se encuentre en ejecución.

> **Sugerencia:** La habilitación de la visibilidad automáticamente habilitará Bluetooth. Si planeas habilitar la visibilidad de dispositivos de manera constante antes de llevar a cabo la actividad de Bluetooth, puedes omitir el paso 2 antes detallado. Lee la sección Habilitación de la visibilidad (#EnablingDiscoverability), a continuación.

Búsqueda de dispositivos

Si usas el `BluetoothAdapter` (https://developer.android.com/reference/android/bluetooth/BluetoothAdapter.html), puedes buscar dispositivos Bluetooth remotos mediante la detección de dispositivos o la consulta de la lista de dispositivos sincronizados (conectados).

La detección de dispositivos es un procedimiento de escaneo que busca en el área local dispositivos con Bluetooth habilitado y, luego, solicita información sobre uno de ellos (comúnmente, esto se denomina "detección", "consulta" o "escaneo"). Sin embargo, un dispositivo Bluetooth dentro del área local solo responderá a la solicitud de visibilidad si actualmente está habilitado para ser detectable. Si un dispositivo es detectable, compartirá información como respuesta a la solicitud de detección, como el nombre del dispositivo, su clase y su dirección MAC única. Al usar esta información, el dispositivo que lleve a cabo la visibilidad podrá iniciar una conexión con el dispositivo detectado.

Una vez que se establezca una conexión con un dispositivo remoto por primera vez, se presentará automáticamente al usuario una solicitud de sincronización. Cuando un dispositivo está sincronizado, la información básica sobre este (como el nombre, la clase y la dirección MAC) se guarda y se puede leer a través de las Bluetooth API. Usando la dirección MAC conocida de un dispositivo remoto, se puede establecer una conexión con este en cualquier momento sin llevar a cabo la detección (suponiendo que el dispositivo se encuentre dentro del rango).

Recuerda que existe una diferencia entre la sincronización y la conexión. Para estar sincronizados, dos dispositivos deben reconocer su existencia mutuamente, tener una clave de vinculación compartida que pueda usarse para autenticación y ser capaces de establecer una conexión encriptada entre sí. Para estar conectados, los dispositivos deben compartir un canal RFCOMM y tener la capacidad de transmitirse datos entre sí. Las Android Bluetooth API actuales requieren que los dispositivos estén sincronizados para que se pueda establecer una conexión RFCOMM. (La sincronización se realiza automáticamente cuando inicies una conexión encriptada con las Bluetooth API).

En las siguientes secciones, se describe la manera de buscar dispositivos sincronizados o hallar dispositivos nuevos mediante la detección de dispositivos.

> **Nota:** Los dispositivos con tecnología de Android no pueden detectarse mediante configuración predeterminada. Un usuario puede hacer que el dispositivo sea detectable durante un tiempo limitado a través de los ajustes del sistema o una aplicación puede solicitar que el usuario habilite

Consulta de dispositivos sincronizados

Antes de llevar a cabo la detección de dispositivos, es importante consultar el conjunto de dispositivos sincronizados a fin de ver si el dispositivo deseado ya es conocido. Para ello, llama a `getBondedDevices()` (https://developer.android.com/reference/android/bluetooth/BluetoothAdapter.html#getBondedDevices()). Esto mostrará un conjunto de `BluetoothDevice` (https://developer.android.com/reference/android/bluetooth/BluetoothDevice.html) que representa los dispositivos sincronizados. Por ejemplo, puedes consultar todos los dispositivos sincronizados y luego mostrar el nombre de cada uno al usuario mediante ArrayAdapter:

```
Set<BluetoothDevice> pairedDevices = mBluetoothAdapter.getBondedDevices();
// If there are paired devices
if (pairedDevices.size() > 0) {
    // Loop through paired devices
    for (BluetoothDevice device : pairedDevices) {
        // Add the name and address to an array adapter to show in a ListView
        mArrayAdapter.add(device.getName() + "\n" + device.getAddress());
    }
}
```

Lo único que se necesita del objeto de `BluetoothDevice` (https://developer.android.com/reference/android/bluetooth/BluetoothDevice.html) para iniciar una conexión es la dirección MAC. En este ejemplo, se guarda como parte de una clase ArrayAdapter que se muestra al usuario. La dirección MAC puede obtenerse en otro momento a fin de establecer la conexión. Puedes obtener más información acerca de cómo crear una conexión en la sección Conexión de dispositivos (#ConnectingDevices).

Detección de dispositivos

Para comenzar a detectar dispositivos, simplemente, llama a `startDiscovery()` (https://developer.android.com/reference/android/bluetooth/BluetoothAdapter.html#startDiscovery()). El proceso es asincrónico y el método mostrará inmediatamente un booleano que indica si la visibilidad se inició con éxito. Por lo general, el proceso de visibilidad incluye un escaneo de consulta de 12 segundos aproximadamente, seguido de un escaneo de la página de cada dispositivo encontrado a fin de recuperar su nombre de Bluetooth.

Tu aplicación debe registrar un BroadcastReceiver para la intent `ACTION_FOUND` (https://developer.android.com/reference/android/bluetooth/BluetoothDevice.html#ACTION_FOUND) a fin de recibir información sobre cada dispositivo detectado. Para cada dispositivo, el sistema transmitirá la intent `ACTION_FOUND` (https://developer.android.com/reference/android/bluetooth/BluetoothDevice.html#ACTION_FOUND). Esta intent incluye los campos extra `EXTRA_DEVICE` (https://developer.android.com/reference/android/bluetooth/BluetoothDevice.html#EXTRA_DEVICE) y `EXTRA_CLASS` (https://developer.android.com/reference/android/bluetooth/BluetoothDevice.html#EXTRA_CLASS), que contienen un `BluetoothDevice` (https://developer.android.com/reference/android/bluetooth/BluetoothDevice.html) y una `BluetoothClass` (https://developer.android.com/reference/android/bluetooth/BluetoothClass.html), respectivamente. Por ejemplo, a continuación se describe cómo puedes registrarte para manejar la transmisión cuando se detectan los dispositivos:

```
// Create a BroadcastReceiver for ACTION_FOUND
private final BroadcastReceiver mReceiver = new BroadcastReceiver() {
    public void onReceive(Context context, Intent intent) {
        String action = intent.getAction();
        // When discovery finds a device
        if (BluetoothDevice.ACTION_FOUND.equals(action)) {
            // Get the BluetoothDevice object from the Intent
            BluetoothDevice device = intent.getParcelableExtra(BluetoothDevice.EXTRA_DEVICE);
            // Add the name and address to an array adapter to show in a ListView
            mArrayAdapter.add(device.getName() + "\n" + device.getAddress());
```

```
// Register the BroadcastReceiver
IntentFilter filter = new IntentFilter(BluetoothDevice.ACTION_FOUND);
registerReceiver(mReceiver, filter); // Don't forget to unregister during onDestroy
```

Lo único que se necesita del objeto de `BluetoothDevice` (https://developer.android.com/reference/android/bluetooth/BluetoothDevice.html) para iniciar una conexión es la dirección MAC. En este ejemplo, se guarda como parte de un ArrayAdapter que se muestra al usuario. La dirección MAC puede obtenerse en otro momento a fin de establecer la conexión. Puedes obtener más información acerca de cómo crear una conexión en la sección Conexión de dispositivos (#ConnectingDevices).

> **Advertencia:** La detección de un dispositivo es un procedimiento complejo para el adaptador de Bluetooth y consumirá gran parte de sus recursos. Una vez que hayas encontrado un dispositivo al cual conectarte, asegúrate de detener la detección con `cancelDiscovery()` (https://developer.android.com/reference/android/bluetooth/BluetoothAdapter.html#cancelDiscovery()) antes de intentar establecer una conexión. Además, si ya dispones de una conexión con un dispositivo, llevar a cabo la detección puede reducir considerablemente el ancho de banda disponible para la conexión, por lo que no debes iniciar la visibilidad mientras estás conectado.

Habilitación de la visibilidad

Si deseas hacer que el dispositivo local sea detectable para otros dispositivos, llama a `startActivityForResult(Intent, int)` (https://developer.android.com/reference/android/app/Activity.html#startActivityForResult(android.content.Intent, int)) con la intent de acción `ACTION_REQUEST_DISCOVERABLE` (https://developer.android.com/reference/android/bluetooth/BluetoothAdapter.html#ACTION_REQUEST_DISCOVERABLE). Esto emitirá una solicitud para habilitar el modo de capacidad de detección mediante la configuración del sistema (sin detener tu aplicación). Según la configuración predeterminada, el dispositivo será detectable durante 120 segundos. Puedes definir una duración diferente si agregando el valor extra de intent `EXTRA_DISCOVERABLE_DURATION` (https://developer.android.com/reference/android/bluetooth/BluetoothAdapter.html#EXTRA_DISCOVERABLE_DURATION). La duración máxima en la que puede fijarse una app es 3600 segundos y un valor igual a 0 significa que el dispositivo siempre es detectable (todo valor inferior a 0 ó superior a 3600 segundos se fija automáticamente en 120). Por ejemplo, este fragmento fija la duración en 300:

```
Intent discoverableIntent = new
Intent(BluetoothAdapter.ACTION_REQUEST_DISCOVERABLE);
discoverableIntent.putExtra(BluetoothAdapter.EXTRA_DISCOVERABLE_DURATION, 300);
startActivity(discoverableIntent);
```

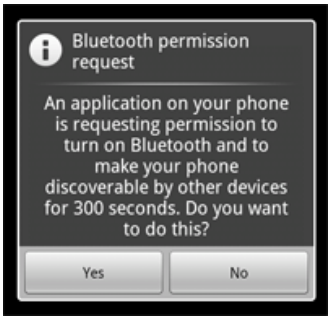

Figura 2: Diálogo de habilitación de la visibilidad.

Se mostrará un diálogo, como en la figura 2, en el cual se solicitará permiso al usuario para hacer que el dispositivo sea detectable, como se muestra en la figura 2. Si el usuario responde “Sí”, el dispositivo será detectable durante el período especificado. Luego, tu actividad recibirá una llamada al callback `onActivityResult())` (https://developer.android.com/reference/android/app/Activity.html#onActivityResult(int, int, android.content.Intent)), con un código de resultado igual a la duración por la cual el dispositivo es detectable. Si el usuario responde “No” o se produce un error, el código de resultado será `RESULT_CANCELED` (https://developer.android.com/reference/android/app/Activity.html#RESULT_CANCELED).

El dispositivo se mantendrá silenciosamente en el modo de capacidad de detección durante el tiempo asignado. Si deseas recibir una notificación cuando se modifique el modo de capacidad de detección, podrás registrar una clase BroadcastReceiver para la intent `ACTION_SCAN_MODE_CHANGED` (https://developer.android.com/reference/android/bluetooth/BluetoothAdapter.html#ACTION_SCAN_MODE_CHANGED). Esto incluirá los campos extra `EXTRA_SCAN_MODE` (https://developer.android.com/reference/android/bluetooth/BluetoothAdapter.html#EXTRA_SCAN_MODE) y `EXTRA_PREVIOUS_SCAN_MODE` (https://developer.android.com/reference/android/bluetooth/BluetoothAdapter.html#EXTRA_PREVIOUS_SCAN_MODE), los cuales te notificarán el modo de escaneo nuevo y el antiguo, respectivamente. Los valores posibles para cada uno son `SCAN_MODE_CONNECTABLE_DISCOVERABLE` (https://developer.android.com/reference/android/bluetooth/BluetoothAdapter.html#SCAN_MODE_CONNECTABLE_DISCOVERABLE), `SCAN_MODE_CONNECTABLE` (https://developer.android.com/reference/android/bluetooth/BluetoothAdapter.html#SCAN_MODE_CONNECTABLE) o `SCAN_MODE_NONE` (https://developer.android.com/reference/android/bluetooth/BluetoothAdapter.html#SCAN_MODE_NONE), que indican si el dispositivo se encuentra en el modo de capacidad de detección, y si no se encuentra en él y puede, o no, recibir conexiones, respectivamente.

No es necesario que habilites la visibilidad de dispositivos si piensas inicializar la conexión con un dispositivo remoto. Solo será necesario habilitarla cuando desees que tu aplicación aloje un socket de servidor que aceptará conexiones entrantes, ya que los dispositivos remotos deben poder detectar el dispositivo antes de inicializar la conexión.

Conexión de dispositivos

A fin de crear una conexión en tu aplicación en dos dispositivos, debes implementar los mecanismos del lado del servidor y del lado del cliente porque un dispositivo debe abrir un socket de servidor y el otro debe inicializar la conexión (usando la dirección MAC del dispositivo del servidor para inicializar la conexión). El servidor y el cliente se consideran conectados entre sí cuando tienen un `BluetoothSocket` (https://developer.android.com/reference/android/bluetooth/BluetoothSocket.html) conectado en el mismo canal RFCOMM. En este punto, cada dispositivo puede obtener flujos de entrada y salida, y se puede iniciar la transferencia de datos, que se trata en la sección Administración de una conexión (#ManagingAConnection). En esta sección, se describe la manera de inicializar la conexión entre dos dispositivos.

El dispositivo del servidor y el dispositivo del cliente obtienen el `BluetoothSocket` (https://developer.android.com/reference/android/bluetooth/BluetoothSocket.html) obligatorio de diferentes maneras. El servidor lo recibirá cuando se acepte una conexión entrante. El cliente, cuando abra un canal RFCOMM hacia el servidor.

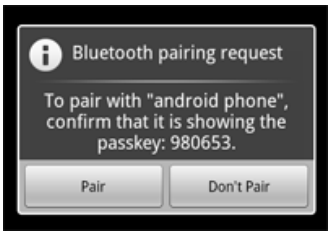

Figura 3: Diálogo de sincronización de Bluetooth.

Una técnica de implementación consiste en preparar automáticamente cada dispositivo como un servidor, de modo que cada uno de ellos tenga un socket de servidor abierto y reciba conexiones. Luego, cualquier dispositivo puede inicializar una conexión con el otro y convertirse en el cliente. Opcionalmente, un dispositivo puede “alojar” de manera explícita la conexión y abrir un socket de servidor a pedido, y el otro dispositivo puede simplemente inicializar la conexión.

> **Nota:** Si los dos dispositivos no se sincronizan previamente, el framework de Android mostrará automáticamente una notificación de solicitud o un diálogo de sincronización al usuario durante el procedimiento de conexión, como se muestra en la Figura 3. De este modo, cuando intente conectar los dispositivos, tu aplicación no debe identificar si los dispositivos están sincronizados o no. Tu intento de conexión RFCOMM se bloqueará hasta que el usuario haya realizado la sincronización con éxito o fallará si el usuario rechaza la sincronización, o en caso de que la sincronización falle o caduque.

Cuando desees conectar dos dispositivos, uno deberá funcionar como servidor y tener un `BluetoothServerSocket` (https://developer.android.com/reference/android/bluetooth/BluetoothServerSocket.html) abierto. El propósito del socket de servidor es recibir solicitudes de conexiones entrantes y, cuando se acepta una, proporcionar un `BluetoothSocket` (https://developer.android.com/reference/android/bluetooth/BluetoothSocket.html) conectado. Cuando el `BluetoothSocket` (https://developer.android.com/reference/android/bluetooth/BluetoothSocket.html) se obtiene del `BluetoothServerSocket` (https://developer.android.com/reference/android/bluetooth/BluetoothServerSocket.html), el `BluetoothServerSocket` (https://developer.android.com/reference/android/bluetooth/BluetoothServerSocket.html) puede (y debe) descartarse, a menos que desees aceptar más conexiones.

A continuación, se describe el procedimiento básico para configurar un socket de servidor y aceptar una conexión:

1. Obtén un `BluetoothServerSocket` (https://developer.android.com/reference/android/bluetooth/BluetoothServerSocket.html); para ello, llama a `listenUsingRfcommWithServiceRecord(String, UUID)` (https://developer.android.com/reference/android/bluetooth/BluetoothAdapter.html#listenUsingRfcommWithServiceRecord(java.lang.String, java.util.UUID)).

   La string es un nombre de identificación de tu servicio, que el sistema escribirá automáticamente en una nueva entrada de la base de datos del protocolo de detección de servicios (SDP) en el dispositivo (el nombre es arbitrario y puede ser simplemente el de tu aplicación). El UUID también se incluye en la entrada del SDP y representará la base del acuerdo de conexión con el dispositivo del cliente. Es decir, cuando el cliente intente conectarse con este dispositivo, se incluirá un UUID que identificará de manera única el servicio con el cual desee conectarse. Estos UUID deben coincidir a fin de que se acepte la conexión (en el paso siguiente).

2. Llama a `accept()` (https://developer.android.com/reference/android/bluetooth/BluetoothServerSocket.html#accept()) para comenzar a recibir solicitudes de conexión.

   Esta es una llamada de bloqueo. Se mostrará cuando se acepte una conexión o se produzca una excepción. Una conexión se acepta solo cuando un dispositivo remoto envía una solicitud de conexión con un UUID que coincide con el UUID registrado con este socket de servidor de recepción. Cuando se realice con éxito, `accept()` (https://developer.android.com/reference/android/bluetooth/BluetoothServerSocket.html#accept()) mostrará un `BluetoothSocket` (https://developer.android.com/reference/android/bluetooth/BluetoothSocket.html) conectado.

3. A menos que desees aceptar conexiones adicionales, llama a `close()` (https://developer.android.com/reference/android/bluetooth/BluetoothServerSocket.html#close()).

   Esto libera el socket de servidor y todos sus recursos, pero *no* cierra el `BluetoothSocket` (https://developer.android.com/reference/android/bluetooth/BluetoothSocket.html) conectado que `accept()` (https://developer.android.com/reference/android/bluetooth/BluetoothServerSocket.html#accept()) muestra. A diferencia del protocolo TCP/IP, RFCOMM solo permite un cliente conectado por canal a la vez, por lo cual en la mayoría de los casos tiene sentido llamar a `close()` (https://developer.android.com/reference/android/bluetooth/BluetoothServerSocket.html#close()) en el `BluetoothServerSocket` (https://developer.android.com/reference/android/bluetooth/BluetoothServerSocket.html) inmediatamente después de aceptar un socket conectado.

Acerca del UUID

Un identificador único universal (UUID) es un formato estandarizado de 128 bits para un ID de string empleado para identificar información de manera exclusiva. La ventaja de un UUID es que es suficientemente grande como para que puedas seleccionar cualquiera al azar y no haya conflictos. En este caso, se usa para identificar de manera única el servicio Bluetooth de tu aplicación. A fin de obtener un UUID que pueda usarse con tu aplicación, puedes emplear uno de los numerosos generadores de UUID al azar de la Web e inicializar un `UUID` (https://developer.android.com/reference/java/util/UUID.html) con `fromString(String)` (https://developer.android.com/reference/java/util/UUID.html#fromString(java.lang.String)).

La llamada a `accept()` (https://developer.android.com/reference/android/bluetooth/BluetoothServerSocket.html#accept()) no debe ejecutarse en el subproceso de la IU de la actividad principal porque es de bloqueo y evitará cualquier otra interacción con la aplicación. Por lo general, tiene sentido hacer todo el trabajo con un `BluetoothServerSocket` (https://developer.android.com/reference/android/bluetooth/BluetoothServerSocket.html) o `BluetoothSocket` (https://developer.android.com/reference/android/bluetooth/BluetoothSocket.html) en un nuevo subproceso administrado por tu aplicación. Para anular una llamada bloqueada, como `accept()` (https://developer.android.com/reference/android/bluetooth/BluetoothServerSocket.html#accept()), llama a `close()`

inmediato. Ten en cuenta que todos los métodos en un `BluetoothServerSocket` (https://developer.android.com/reference/android/bluetooth/BluetoothServerSocket.html) o `BluetoothSocket` (https://developer.android.com/reference/android/bluetooth/BluetoothSocket.html) tienen seguridad para subprocesos.

Ejemplo

A continuación, se describe un subproceso simplificado para el componente de servidor que acepta conexiones entrantes:

```
private class AcceptThread extends Thread {
    private final BluetoothServerSocket mmServerSocket;

    public AcceptThread() {
        // Use a temporary object that is later assigned to mmServerSocket,
        // because mmServerSocket is final
        BluetoothServerSocket tmp = null;
        try {
            // MY_UUID is the app's UUID string, also used by the client code
            tmp = mBluetoothAdapter.listenUsingRfcommWithServiceRecord(NAME, MY_UUID);
        } catch (IOException e) { }
        mmServerSocket = tmp;
    }

    public void run() {
        BluetoothSocket socket = null;
        // Keep listening until exception occurs or a socket is returned
        while (true) {
            try {
                socket = mmServerSocket.accept();
            } catch (IOException e) {
                break;
            }
            // If a connection was accepted
            if (socket != null) {
                // Do work to manage the connection (in a separate thread)
                manageConnectedSocket(socket);
                mmServerSocket.close();
                break;
            }
        }
    }

    /** Will cancel the listening socket, and cause the thread to finish */
    public void cancel() {
        try {
            mmServerSocket.close();
        } catch (IOException e) { }
    }
}
```

En este ejemplo, solo se desea una conexión entrante; por lo tanto, apenas se acepta una conexión y se obtiene el `BluetoothSocket` (https://developer.android.com/reference/android/bluetooth/BluetoothSocket.html), la aplicación envía el `BluetoothSocket` (https://developer.android.com/reference/android/bluetooth/BluetoothSocket.html) adquirido a un subproceso separado, cierra el `BluetoothServerSocket` (https://developer.android.com/reference/android/bluetooth/BluetoothServerSocket.html) e interrumpe el bucle.

Ten en cuenta que, cuando `accept()` (https://developer.android.com/reference/android/bluetooth/BluetoothServerSocket.html#accept()) muestra el `BluetoothSocket` (https://developer.android.com/reference/android/bluetooth/BluetoothSocket.html), el socket ya se encuentra conectado, por lo cual *no* debes llamar a `connect()` (https://developer.android.com/reference/android/bluetooth/BluetoothSocket.html#connect()) (como lo haces del del lado del cliente).

Por lo general, debes cerrar tu `BluetoothServerSocket` (https://developer.android.com/reference/android/bluetooth/BluetoothServerSocket.html) no bien termines de recibir conexiones entrantes. En este ejemplo, se llama a `close()` (https://developer.android.com/reference/android/bluetooth/BluetoothServerSocket.html#close()) apenas se obtiene el `BluetoothSocket` (https://developer.android.com/reference/android/bluetooth/BluetoothSocket.html). Es posible que también desees proporcionar en tu subproceso un método público que pueda cerrar el `BluetoothSocket` (https://developer.android.com/reference/android/bluetooth/BluetoothSocket.html) en caso de que necesites dejar de recibir conexiones en el socket de servidor.

Conexión como cliente

A fin de inicializar una conexión con un dispositivo remoto (un dispositivo que mantenga un socket de servidor abierto), primero debes obtener un objeto `BluetoothDevice` (https://developer.android.com/reference/android/bluetooth/BluetoothDevice.html) que represente el dispositivo remoto. (La obtención de un `BluetoothDevice` (https://developer.android.com/reference/android/bluetooth/BluetoothDevice.html) se trata en la sección anterior, Búsqueda de dispositivos (#FindingDevices)). Luego, debes usar el `BluetoothDevice` (https://developer.android.com/reference/android/bluetooth/BluetoothDevice.html) para obtener un `BluetoothSocket` (https://developer.android.com/reference/android/bluetooth/BluetoothSocket.html) e inicializar la conexión.

A continuación, se describe el procedimiento básico:

1. Usando el `BluetoothDevice` (https://developer.android.com/reference/android/bluetooth/BluetoothDevice.html), obtén un `BluetoothSocket` (https://developer.android.com/reference/android/bluetooth/BluetoothSocket.html); para ello, llama a `createRfcommSocketToServiceRecord(UUID)` (https://developer.android.com/reference/android/bluetooth/BluetoothDevice.html#createRfcommSocketToServiceRecord(java.util.UUID)).

   Esto inicializa un `BluetoothSocket` (https://developer.android.com/reference/android/bluetooth/BluetoothSocket.html) que se conectará al `BluetoothDevice` (https://developer.android.com/reference/android/bluetooth/BluetoothDevice.html). El UUID pasado aquí debe coincidir con el UUID empleado por el dispositivo del servidor cuando abrió su `BluetoothServerSocket` (https://developer.android.com/reference/android/bluetooth/BluetoothServerSocket.html) (con `listenUsingRfcommWithServiceRecord(String, UUID)` (https://developer.android.com/reference/android/bluetooth/BluetoothAdapter.html#listenUsingRfcommWithServiceRecord(java.lang.String, java.util.UUID))). Usar el mismo UUID implica simplemente codificar de manera rígida la string del UUID en tu aplicación y luego hacer referencia a este desde el código del cliente y el servidor.

2. Llama a `connect()` (https://developer.android.com/reference/android/bluetooth/BluetoothSocket.html#connect()) para inicializar la conexión.

   Una vez que se realice esta llamada, el sistema realizará una búsqueda del SDP en el dispositivo remoto a fin de que coincida el UUID. Si la búsqueda tiene éxito y el dispositivo remoto acepta la conexión, este último compartirá el canal RFCOMM que se usará durante la conexión y se mostrará `connect()` (https://developer.android.com/reference/android/bluetooth/BluetoothSocket.html#connect()). Este método es una llamada de bloqueo. Si, por algún motivo, la conexión falla o el método `connect()` (https://developer.android.com/reference/android/bluetooth/BluetoothSocket.html#connect()) caduca (después de unos 12 segundos), se generará una excepción.

   Debido a que `connect()` (https://developer.android.com/reference/android/bluetooth/BluetoothSocket.html#connect()) es una llamada de bloqueo, este procedimiento de conexión siempre debe realizarse en un proceso por separado del subproceso de la actividad principal.

> Nota: Siempre debes asegurarte de que el dispositivo no realice la detección de dispositivos cuando llamas a `connect()` (https://developer.android.com/reference/android/bluetooth/BluetoothSocket.html#connect()). Si la detección se encuentra en curso, entonces el intento de conexión se ralentizará considerablemente y será más probable que falle.

Ejemplo

A continuación, se ofrece un ejemplo básico de un subproceso que inicializa una conexión Bluetooth:

```
private class ConnectThread extends Thread {
    private final BluetoothSocket mmSocket;
    private final BluetoothDevice mmDevice;
```

```
        BluetoothSocket tmp = null;
        mmDevice = device;

        // Get a BluetoothSocket to connect with the given BluetoothDevice
        try {
            // MY_UUID is the app's UUID string, also used by the server code
            tmp = device.createRfcommSocketToServiceRecord(MY_UUID);
        } catch (IOException e) { }
        mmSocket = tmp;
    }

    public void run() {
        // Cancel discovery because it will slow down the connection
        mBluetoothAdapter.cancelDiscovery();

        try {
            // Connect the device through the socket. This will block
            // until it succeeds or throws an exception
            mmSocket.connect();
        } catch (IOException connectException) {
            // Unable to connect; close the socket and get out
            try {
                mmSocket.close();
            } catch (IOException closeException) { }
            return;
        }

        // Do work to manage the connection (in a separate thread)
        manageConnectedSocket(mmSocket);
    }

    /** Will cancel an in-progress connection, and close the socket */
    public void cancel() {
        try {
            mmSocket.close();
        } catch (IOException e) { }
    }
}
```

Ten en cuenta que se llama a `cancelDiscovery()` (https://developer.android.com/reference/android/bluetooth/BluetoothAdapter.html#cancelDiscovery()) antes de que se realice la conexión. Siempre debes hacer esto antes de la conexión, y resultará seguro realizar la llamada sin verificar realmente si está en ejecución o no (si quieres esta verificación, no obstante, llama a `isDiscovering()` (https://developer.android.com/reference/android/bluetooth/BluetoothAdapter.html#isDiscovering())).

`manageConnectedSocket()` es un método de ficción en la aplicación que inicializará el subproceso para transferir datos, lo cual se analiza en la sección Administración de una conexión (#ManagingAConnection).

Cuando finalices con tu `BluetoothSocket` (https://developer.android.com/reference/android/bluetooth/BluetoothSocket.html), siempre llama a `close()` (https://developer.android.com/reference/android/bluetooth/BluetoothSocket.html#close()) para hacer una limpieza. Cuando lo hagas, se cerrará de inmediato el socket conectado y se limpiarán todos los recursos internos.

Administración de una conexión

Cuando hayas conectado con éxito dos (o más) dispositivos, cada uno tendrá un `BluetoothSocket` (https://developer.android.com/reference/android/bluetooth/BluetoothSocket.html) conectado. Aquí es donde comienza la diversión porque puedes compartir datos entre dispositivos. Mediante el `BluetoothSocket` (https://developer.android.com/reference/android/bluetooth/BluetoothSocket.html), el procedimiento general para transferir datos arbitrarios es simple:

1. Obtén el `InputStream` (https://developer.android.com/reference/java/io/InputStream.html) y `OutputStream` (https://developer.android.com/reference/java/io/OutputStream.html) que administran transmisiones a través del socket, mediante `getInputStream()` (https://developer.android.com/reference/android/bluetooth/BluetoothSocket.html#getInputStream()) y `getOutputStream()` (https://developer.android.com/reference/android/bluetooth/BluetoothSocket.html#getOutputStream()), respectivamente.
2. Lee y escribe datos para los flujos con `read(byte[])` (https://developer.android.com/reference/java/io/InputStream.html#read(byte[])) y `write(byte[])` (https://developer.android.com/reference/java/io/OutputStream.html#write(byte[])).

Eso es todo.

Por supuesto, existen detalles relacionados con la implementación que deben tenerse en cuenta. En primer lugar, debes usar un subproceso dedicado para toda lectura y escritura de flujos. Esto es importante porque los métodos `read(byte[])` (https://developer.android.com/reference/java/io/InputStream.html#read(byte[])) y `write(byte[])` (https://developer.android.com/reference/java/io/OutputStream.html#write(byte[])) son llamadas de bloqueo. `read(byte[])` (https://developer.android.com/reference/java/io/InputStream.html#read(byte[])) aplicará un bloqueo hasta que exista algo para leer del flujo. Por lo general, `write(byte[])` (https://developer.android.com/reference/java/io/OutputStream.html#write(byte[])) no aplica bloqueos, pero puede hacerlo a los fines de controlar el flujo si el dispositivo remoto no llama a `read(byte[])` (https://developer.android.com/reference/java/io/InputStream.html#read(byte[])) con suficiente rapidez y los búferes intermedios están completos. Por lo tanto, tu bucle principal del subproceso debe ser dedicado para la lectura desde el `InputStream` (https://developer.android.com/reference/java/io/InputStream.html). Se puede usar un método público por separado en el subproceso a fin de iniciar escrituras en el `OutputStream` (https://developer.android.com/reference/java/io/OutputStream.html).

Ejemplo

A continuación, se ofrece un ejemplo de cómo podría ser esto:

```
private class ConnectedThread extends Thread {
    private final BluetoothSocket mmSocket;
    private final InputStream mmInStream;
    private final OutputStream mmOutStream;

    public ConnectedThread(BluetoothSocket socket) {
        mmSocket = socket;
        InputStream tmpIn = null;
        OutputStream tmpOut = null;

        // Get the input and output streams, using temp objects because
        // member streams are final
        try {
            tmpIn = socket.getInputStream();
            tmpOut = socket.getOutputStream();
        } catch (IOException e) { }

        mmInStream = tmpIn;
        mmOutStream = tmpOut;
    }

    public void run() {
        byte[] buffer = new byte[1024];  // buffer store for the stream
        int bytes; // bytes returned from read()

        // Keep listening to the InputStream until an exception occurs
        while (true) {
            try {
                // Read from the InputStream
                bytes = mmInStream.read(buffer);
                // Send the obtained bytes to the UI activity
                mHandler.obtainMessage(MESSAGE_READ, bytes, -1, buffer)
```

```
            }
        }
    }

    /* Call this from the main activity to send data to the remote device */
    public void write(byte[] bytes) {
        try {
            mmOutStream.write(bytes);
        } catch (IOException e) { }
    }

    /* Call this from the main activity to shutdown the connection */
    public void cancel() {
        try {
            mmSocket.close();
        } catch (IOException e) { }
    }
}
```

El constructor obtiene los flujos necesarios y, una vez ejecutados estos, el subproceso esperará el ingreso de datos mediante InputStream. Cuando `read(byte[])` (https://developer.android.com/reference/java/io/InputStream.html#read(byte[])) muestra bytes del flujo, los datos se envían a la actividad principal mediante un controlador de un miembro de la clase primaria. Luego, regresa y espera más del flujo.

El envío de datos salientes simplemente implica llamar al método `write()` del subproceso desde la actividad principal y pasar los bytes que se enviarán. Este método simplemente llama a `write(byte[])` (https://developer.android.com/reference/java/io/OutputStream.html#write(byte[])) para enviar los datos al dispositivo remoto.

El método `cancel()` del subproceso es importante para que la conexión se pueda finalizar en cualquier momento cerrando `BluetoothSocket` (https://developer.android.com/reference/android/bluetooth/BluetoothSocket.html). Siempre debe llamarse a esto cuando termines de usar la conexión Bluetooth.

Para ver una demostración de uso de las Bluetooth API, consulta el ejemplo de app Chat de Bluetooth (https://developer.android.com/resources/samples/BluetoothChat/index.html).

Trabajo con perfiles

A partir de Android 3.0, la Bluetooth API incluye compatibilidad para trabajar con perfiles de Bluetooth. Un *perfil de Bluetooth* es una especificación de interfaz inalámbrica para la comunicación entre dispositivos basada en Bluetooth. Un ejemplo es el perfil de manos libres. Para que un teléfono móvil se conecte a auriculares inalámbricos, ambos dispositivos deben ser compatibles con el perfil de manos libres.

Puedes implementar la interfaz `BluetoothProfile` (https://developer.android.com/reference/android/bluetooth/BluetoothProfile.html) para escribir tus propias clases a fin de que admitan un perfil de Bluetooth en particular. La Android Bluetooth API proporciona implementaciones para los siguientes perfiles de Bluetooth:

- **Auriculares**: el perfil de auriculares ofrece compatibilidad con auriculares Bluetooth para usarlos con teléfonos móviles. Android proporciona la clase `BluetoothHeadset` (https://developer.android.com/reference/android/bluetooth/BluetoothHeadset.html), que es un proxy para controlar el servicio de auriculares Bluetooth a través de la comunicación entre procesos (IPC (https://developer.android.com/guide/components/processes-and-threads.html#IPC)). Esto incluye los perfiles de manos libres (v1.5) y auriculares Bluetooth. La clase `BluetoothHeadset` (https://developer.android.com/reference/android/bluetooth/BluetoothHeadset.html) incluye compatibilidad para los comandos de AT. Para obtener más información sobre este tema, consulta la sección Comandos de AT específicos del proveedor (#AT-Commands).
- **A2DP**: el perfil de distribución de audio avanzada (A2DP) define el nivel de audio de alta calidad que puede transmitirse de un dispositivo a otro a través de una conexión Bluetooth. Android ofrece la clase `BluetoothA2dp`

- **Dispositivo de salud**: Android 4.0 (nivel de API 14) presenta compatibilidad con el perfil de dispositivos de salud (HDP) Bluetooth. Esto te permite crear aplicaciones que usan Bluetooth para comunicarse con dispositivos de salud que admiten Bluetooth, como monitores de frecuencia cardíaca, medidores de la sangre, termómetros y balanzas, entre otros. Para acceder a una lista de los dispositivos compatibles y sus códigos de especialización correspondientes de los datos del dispositivo, consulta la sección **Números asignados de Bluetooth** en www.bluetooth.org (http://www.bluetooth.org). Ten en cuenta que también se hace referencia a estos valores en la especificación ISO/IEEE 11073-20601 [7] como MDC_DEV_SPEC_PROFILE_* en el Anexo de códigos de nomenclaturas. Para obtener más información sobre el HDP, consulta la sección Perfil de dispositivos de salud (#HDP).

A continuación, se ofrecen los pasos básicos para trabajar con un perfil:

1. Obtén el adaptador predeterminado, tal como se describe en la sección Configuración de Bluetooth (https://developer.android.com/guide/topics/connectivity/bluetooth.html#SettingUp).
2. Usa `getProfileProxy()` (https://developer.android.com/reference/android/bluetooth/BluetoothAdapter.html#getProfileProxy(android.content.Context, android.bluetooth.BluetoothProfile.ServiceListener, int)) para establecer una conexión con el objeto del proxy de perfil asociado al perfil en cuestión. En el siguiente ejemplo, el objeto del proxy de perfil es una instancia de `BluetoothHeadset` (https://developer.android.com/reference/android/bluetooth/BluetoothHeadset.html).
3. Configura un `BluetoothProfile.ServiceListener` (https://developer.android.com/reference/android/bluetooth/BluetoothProfile.ServiceListener.html). Este notifica a los clientes de IPC del `BluetoothProfile` (https://developer.android.com/reference/android/bluetooth/BluetoothProfile.html) cuando se conectan al servicio o se desconectan de este.
4. En `onServiceConnected()` (https://developer.android.com/reference/android/bluetooth/BluetoothProfile.ServiceListener.html#onServiceConnected(int, android.bluetooth.BluetoothProfile)), obtén un controlador para el objeto del proxy de perfil.
5. Una vez que tengas el objeto del proxy de perfil, puedes usarlo para controlar el estado de la conexión y realizar otras operaciones pertinentes para ese perfil.

Por ejemplo, en este fragmento de código se muestra la manera de conectarse a un objeto del proxy de `BluetoothHeadset` (https://developer.android.com/reference/android/bluetooth/BluetoothHeadset.html) de modo que se pueda controlar el perfil de auriculares:

```
BluetoothHeadset mBluetoothHeadset;

// Get the default adapter
BluetoothAdapter mBluetoothAdapter = BluetoothAdapter.getDefaultAdapter();

// Establish connection to the proxy.
mBluetoothAdapter.getProfileProxy(context, mProfileListener, BluetoothProfile.HEADSET);

private BluetoothProfile.ServiceListener mProfileListener = new BluetoothProfile.ServiceListener() {
    public void onServiceConnected(int profile, BluetoothProfile proxy) {
        if (profile == BluetoothProfile.HEADSET) {
            mBluetoothHeadset = (BluetoothHeadset) proxy;
        }
    }
    public void onServiceDisconnected(int profile) {
        if (profile == BluetoothProfile.HEADSET) {
            mBluetoothHeadset = null;
        }
    }
};

// ... call functions on mBluetoothHeadset

// Close proxy connection after use.
mBluetoothAdapter.closeProfileProxy(mBluetoothHeadset);
```

A partir de Android 3.0, las aplicaciones pueden registrarse para recibir transmisiones del sistema de comandos de AT específicos del proveedor previamente definidos que envían los auriculares (como un comando +XEVENT de Plantronics). Por ejemplo, una aplicación podría recibir transmisiones que indiquen el nivel de batería de un dispositivo conectado y notificar al usuario o tomar otras medidas, según sea necesario. Crea un receptor de transmisiones para que la intent `ACTION_VENDOR_SPECIFIC_HEADSET_EVENT` (https://developer.android.com/reference/android/bluetooth/BluetoothHeadset.html#ACTION_VENDOR_SPECIFIC_HEADSET_EVENT) controle comandos de AT específicos del proveedor para los auriculares.

Perfil de dispositivos de salud

Android 4.0 (nivel de API 14) presenta compatibilidad con perfil de dispositivos de salud (HDP) Bluetooth. Esto te permite crear aplicaciones que usan Bluetooth para comunicarse con dispositivos de salud que admiten Bluetooth, como monitores de frecuencia cardíaca, medidores de la sangre, termómetros y balanzas. La Bluetooth Health API incluye las clases `BluetoothHealth` (https://developer.android.com/reference/android/bluetooth/BluetoothHealth.html), `BluetoothHealthCallback` (https://developer.android.com/reference/android/bluetooth/BluetoothHealthCallback.html) y `BluetoothHealthAppConfiguration` (https://developer.android.com/reference/android/bluetooth/BluetoothHealthAppConfiguration.html), que se describen en la sección Aspectos básicos (#TheBasics).

Cuando se usa la Bluetooth Health API, resulta útil comprender estos conceptos claves del HDP:

Concepto	Descripción
Fuente	Rol definido en el HDP. Una *fuente* es un dispositivo de salud que transmite datos médicos (balanza, medidor de glucosa, termómetro, etc.) a un dispositivo inteligente, como un teléfono o una tablet Android.
Receptor	Rol definido en el HDP. En el HDP, un *receptor* es el dispositivo inteligente que recibe los datos médicos. En una aplicación del HDP de Android, el receptor se representa mediante un objeto `BluetoothHealthAppConfiguration` (https://developer.android.com/reference/android/bluetooth/BluetoothHealthAppConfiguration.html).
Registro	Hace referencia al registro de un receptor de un dispositivo de salud en particular.
Conexión	Hace referencia a la apertura de una canal entre un dispositivo de salud y un dispositivo inteligente, como un teléfono o una tablet Android.

Creación de una aplicación del HDP

Estos son los pasos básicos para crear una aplicación del HDP de Android:

1. Obtén una referencia al objeto del proxy `BluetoothHealth` (https://developer.android.com/reference/android/bluetooth/BluetoothHealth.html).

   Aplicando un procedimiento similar al de los dispositivos de perfil A2DP y de auriculares, debes llamar a `getProfileProxy()` (https://developer.android.com/reference/android/bluetooth/BluetoothAdapter.html#getProfileProxy(android.content.Context, android.bluetooth.BluetoothProfile.ServiceListener, int)) con un `BluetoothProfile.ServiceListener` (https://developer.android.com/reference/android/bluetooth/BluetoothProfile.ServiceListener.html) y el tipo de perfil `HEALTH` (https://developer.android.com/reference/android/bluetooth/BluetoothProfile.html#HEALTH) para para establecer una conexión con el objeto del proxy de perfil.

2. Crea un `BluetoothHealthCallback` (https://developer.android.com/reference/android/bluetooth/BluetoothHealthCallback.html) y registra una configuración de la aplicación (`BluetoothHealthAppConfiguration` (https://developer.android.com/reference/android/bluetooth/BluetoothHealthAppConfiguration.html)) que funcione como receptor de salud.

3. Establece una conexión con el dispositivo de salud. Algunos dispositivos iniciarán la conexión. Es necesario seguir este paso en el caso de esos dispositivos.

4. Cuando te conectes con éxito a un dispositivo de salud, realiza operaciones de lectura o escritura en el dispositivo de salud mediante el descriptor del archivo.

5. Cuando finalices, cierra el canal de salud y elimina el registro de la aplicación. El canal también se cierra cuando existe inactividad por tiempo prolongado.

Para acceder a una muestra de códigos completa en la que se ilustren estos pasos, consulta la sección HDP (perfil de dispositivos de salud) Bluetooth (https://developer.android.com/resources/samples/BluetoothHDP/index.html).