Usando o Bluetooth no Android (Tradução da documentação oficial)

A plataforma Android inclui suporte para a pilha da rede Bluetooth, que permite que um dispositivo troque dados sem necessidade de cabos com outros dispositivos Bluetooth. O framework da aplicação fornece acesso a fuincionalidade Bluetooth através das APIs Bluetooth do Android. Essas APIs permitem que a aplicação conecte-se a outros dispositivos Bluetooth, ativando recursos de rede sem fio ponto a ponto e multiponto.

O Básico

Esse artigo descreve como usar as APIs Bluetooth do Android para cumprir as quatro principais tarefas necessárias para a comunicação via Bluetooth: configurar o Bluetooth, encontrar dispositivos que estejam emparelhados ou disponíveis na área, conectar os dispositivos e transferir dados entre os dispositivos.

Todas as APIs Bluetooth estão disponíveis no pacote android.bluetooth. Abaixo segue um resumo das classes e interfaces que você precisará para criar conexões Bluetooth:

BluetoothAdapter

> Representa o adaptador Bluetooth local (radio Bluetooth). OBluetoothAdapter é o ponto de entrada para todas as interações Bluetooth. Usando ele,você pode descobrir outros dispositivos Bluetooth, consultar uma lista de dispositivos emparelhados, instanciar um BluetoothDevice usando um endereço MAC

conhecido, e criar um BluetoothServerSocket para escutar por comunicações de outros dispositivos.

BluetoothDevice

Representa um dispositivo remoto Bluetooth. Use isso para requerer uma conexão com um dispositivo remoto através de um BluetoothSocket ou consultar informações sobre o dispositivo como seu nome, endereço, e estado da ligação.

BluetoothSocket

Representa a interface para um socket Bluetooth (similar a umSocket `TCP`). Ele é um ponto de conexão que permite que uma aplicação troque dados com outro dispositivo Bluetooth através do InputStream e OutputStream.

BluetoothServerSocket

Representa um socket de servidor que escuta por requisições que chegam ao dispositivo onde a aplicação roda (similar a um ServerSocket `TCP`). De forma a conectar dois dispositivos Android, um dos dispositivos precosa abrir um socket servidor com essa classe. Quando um dispositivo Bluetooth faz uma requisição de conexão a esse dispositivo, o BluetoothServerSocket retornará um BluetoothSocket conectado quando a conexão é aceita.

BluetoothClass

Descreve as características gerais e recursos de um dispositivo Bluetooth. Isto é um conjunto de propriedades somente leitura que define todas as classes de dispositivos e seus serviços. Porém, ele não descreve com precisão descreve todos os perfis Bluetooth suportados pelo dispositivo, mas é útil como uma dica do tipo do dispositivo.

BluetoothProfile

Uma interface que representa um perfil Bluetooh. Um *perfil Bluetooth* é uma especificação de uma interface sem fio para comunicação baseada em Bluetooth entre dispositivos. Um exemplo é o perfil *Hands-Free*.

BluetoothHeadset

Fornece suporte para headsets Bluetooth para ser usados em telefones celulares. Inclui tanto Perfis Headset Bluetooth quanto Hands-Free (v1.5).

BluetoothA2dp

Define como áudio de alta qualidade pode ser enviado de um dispositivo para outro sobre uma conexão Bluetooth. “A2DP” significa “Advanced Audio Distribution Profile”.

BluetoothHealth

Representa um Perfil de dispositivo de saúde que controla o serviço Bluetooth.

BluetoothHealthCallback

Uma classe abstrata que você usa para implementar chamadas a BluetoothHealth. Você precisa estender essa classe e implementar os métodos correspondentes para receber atualizações sobre mudanças no estado do registro da aplicação e no estado do canal Bluetooth.

BluetoothProfile.ServiceListener

Uma interface que notifica clientes IPC BluetoothProfile quando eles tem que se conectar ou disconectar (isso é. o serviço interno que roda um perfil em particular).

Permissões do Bluetooth

De modo a usar recursos Bluetooth em sua aplicação, você precisa declarar ao menos uma dessas duas permissões do Bluetooth: BLUETOOTH e BLUETOOTH_ADMIN.
Você precisa requisitar a permissão BLUETOOTH para poder executar qualquer comunicação via Bluetooth, como requisitar uma conexão, aceitar uma conexão e transferir dados.
Você precisa requisitar a permissão BLUETOOTH_ADMIN para poder iniciar a descoberta de dispositivos ou manipular as configurações do Bluetooth. Muitas aplicações precisam dessa permissão somente para ter a habilidade de dscobrir dispositivos Bluetooth. As outras habilidades garantidas por essa permissão não deveriam ser usadas, a menos que a aplicação seja um gerenciador que irá modificar as configurações do Bluetooth baseado no pedido do usuário. **Nota:** Se você usar a permissão BLUETOOTH_ADMIN, precisa ter a permissão BLUETOOTH.
Declare as permissões Bluetooth no seu arquivo de manifesto. Por exemplo:

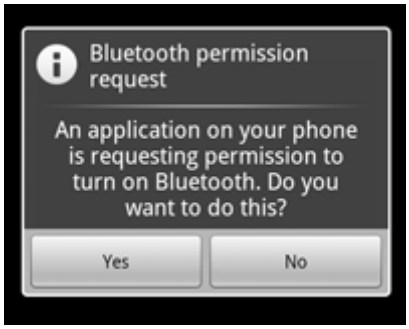
```
<manifest ... >

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

  ...

</manifest>
```

Configurando o Bluetooth



Antes que a sua aplicação possa se comunicar por Bluetooth, é preciso que o Bluetooth seja suportado pelo dispositivo, e nesse caso, é preciso se certificar que esteja ativado.

Se o Bluetooth não for suportado, então você deve graciosamente desativar todos os recursos dependentes dele. Se for suportado, mas estiver desativado, então você pode requerer que o usuário ative o Bluetooth sem sair da aplicação. Esse processo é feito em dois passos, usando o BluetoothAdapter.

Obtenha o BluetoothAdapter

O BluetoothAdapter é necessário para qualquer atividade Bluetooth. Para obter o BluetoothAdapter, chame o método estático getDefaultAdapter(). Esse método retorna um BluetoothAdapter que representa o próprio adaptador Bluetooth do dispositivo. Existe um único adaptador Bluetooth para todo o sistema, e sua aplicação pode interagir com ele usando esse objeto. Se o método getDefaultAdapter() retornar*null*, então o dispositivo não suporta Bluetooth e sua estória termina aqui. Por exemplo:

```
BluetoothAdapter mBluetoothAdapter =
BluetoothAdapter.getDefaultAdapter();

if (mBluetoothAdapter == null) {
```

```
    // Device does not support Bluetooth

}
```

Ative o Bluetooth

Em seguida, você precisa se certificar que o Bluetooth esteja ativado. Chame isEnabled() para checar e o Bluetooth está ativo no momento. Se esse método retornar falso, o Bluetooth está desativado. Para solicitar a ativação do Bluetooth, chame o método startActivityForResult() com o *intent* de ação ACTION_REQUEST_ENABLE. Isso irá disparar uma requisição para ativar o Bluetooth através das configurações do sistema (sem interromper a sua aplicação). Por exemplo:

```
if (!mBluetoothAdapter.isEnabled()) {

    Intent enableBtIntent = new
Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);

    startActivityForResult(enableBtIntent, REQUEST_ENABLE_BT);

}
```

Uma caixa de diálogo irá aparecer solicitando ao usuário permissão para ativar o Bluetooth, como mostrado da Figura 1. Se o usuário reponder *Yes* o sistema iniciará a ativação do Bluetooth e o foco retornará para a sua aplicação uma vez que o processo esteja completo (ou falhe). A constante `REQUEST_ENABLE_BT` passada para o método startActivityForResult() é um inteiro definido localmente (que precisa ser maior do que 0) que o sistema retorna de volta para você em sua implementação de onActivityResult() como o parâmetro `requestCode`. Se a ativação do Bluetooth for bem sucessida, sua Activity recebe o código de resultado RESULT_OK na chamada a onActivityResult(). Se o Bluetooth não for ativado devido a

um erro (ou uma resposta *No* do usuário), o código de resultado será RESULT_CANCELED.
Opcionalmente, sua aplicação pode escutar também pelo *intent de broadcast* ACTION_STATE_CHANGED, por onde o sistema irá informar se o estado do Bluetooth foi alterado. Essa transmissão contém os campos extras EXTRA_STATE e EXTRA_PREVIOUS_STATE, contendo o novo estado e o anterior, respectivamente. Valores possíveis para esses campos extras
são STATE_TURNING_ON, STATE_ON, STATE_TURNING_OFF, e STATE_OFF. Escutar essa transmissão pode ser útil para detectar mudanças de modo feitas ao Bluetooth enquanto a aplicação estiver sendo executada.

Procurando dispositivos

Usando o BluetoothAdapter, você pode procurar por dispositivos Bluetooth remotos seja através da pesquisa de dispositivos ou pela consulta a lista de dispositivos emparelhados.
A pesquisa de dispositivos é o procedimento de escaneamento que procura na área local por dispositivos que estejam com o Bluetooth ativado e solicita alguma informação sobre cada um deles. Porém, um dispositivo Bluetooth dentro de uma determinada área só irá responder a um pedido desse tipo no caso de ter ativado o modo de descoberta. Se um dispositivo estiver nesse modo, responderá a pedidos de pesquisa através do compartilhamento de algumas informações, como nome do dispositivo, classe e seu endereço MAC único. Usando essa informação,

o dispositivo que está executando a busca pode decidir por iniciar um conexão com o dispositivo descoberto.

Uma vez que a conexão for feita pela primeira vez, um pedido de emparelhamento é automaticamente apresentado ao usuário. Quando um dispositivo é emparelhado as informações básicas sobre esse dispositivo (como nome do dispositivo, classe, endereço MAC) é salvo e pode ser lido usando as APIs Bluetooth. Usando o endereço MAC conhecido do dispositivo remoto, uma conexão pode ser iniciada a qualquer tempo sem necessidade de ser feita uma pesquisa (assumindo que o dispositivo esteja ao alcance).

Lembre que existe uma diferença em estar emparelhado e estar conectado. Estar emparelhado significa que os dois dispositivos estão cientes da existência um do outro, possuem uma chave de ligação compartilhada que pode ser usada para autenticação, e são capazes de estabelecer uma conexão encriptada entre eles. Estar conectado significa que os dispositivos no momento compartilham uma canal RFCOMM e são capazes de transmitir dados entre eles. A API Bluetooth do Android requer que os dispositivos sejam emparelhados antes que uma conexão RFCOMM possa ser estabelecida (o emparelhamento é automaticamente executado quando você inicia uma conexão criptografada com a API Bluetooth).

Consultando dispositivos emparelhados

Antes de executar uma pesquisa por dispositivos, vale a pena checar a lista de dispositivos emparelhados para ver se o dispositivo desejado já é conhecido. Para fazer isso, chame getBondedDevices(). Esse método retornará um Set de BluetoothDevice's que representa os dispositivos

emparelhados. Por exemplo, você pode consultar todos os dispositivos emparelhados e exibir o nome de cada dispositivo ao usuário usando um 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());

    }

}
```

Tudo o que é necessário do objeto BluetoothDevice para iniciar uma conexão é o endereço MAC. Nesse exemplo, é salvo como uma parte de um ArrayAdapter que é exibido ao usuário. O endereço MAC pode mais tarde ser extraído para iniciar a conexão.

Descobrindo dispositivos

Para iniciar a descoberta de dispositivos, simplesmente chame startDiscovery(). O processo é assíncrono e o método imediatamente retornará um booleano indicando se a descoberta foi iniciada com sucesso. O processo de descoberta usualmente envolve um escaneamento inquisitivo que dura aproximadamente 12 segundos,

seguido por um escaneamento de cada dispositivo encontrado para recuperar o nome dele.

Sua aplicação precisa registrar o *intent* ACTION_FOUND de forma a poder receber informações sobre cada dispositivo encontrado. Para cada dispositivo, o sistema irá transmitir o *intent* ACTION_FOUND.

Esse *intent* carrega os campos extras EXTRA_DEVICE e EXTRA_CLASS, que contém o BluetoothDevice e um BluetoothClass, respectivamente. Por exemplo, abaixo segue como você pode se registrar para manipular com a transmissão quando os dispositivos são descobertos:

```
// 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
```

Cuidado: Executar esse tipo de pesuisa por dispositivos é um procedimento pesado para o adaptador Bluetooth e consumirá muitos recursos. Uma vez que você tenha encontrado um dispositivo para conectar, certifique-se de interromper o processo de pesquisa com cancelDiscovery() antes de tentar uma conexão. Além disso, se você já possuir uma conexão com um dispositivo, executar uma pesquisa por dispositivos pode reduzir significativamente a banda disponível para a conexão, de forma que você não deve executar uma pesquisa enquanto estiver conectado.

Permitindo a descoberta do dispositivo

Se você quiser tornar o dispositivo onde a aplicação está rodando disponível para que outros dispositivos o achem, chame startActivityForResult(Intent, int) com o *intent* de ação ACTION_REQUEST_DISCOVERABLE. Isso irá disparar uma requisição para ativar o modo de descoberta pelo sistema (sem interromper a sua aplicação). Por padrão, o dispositivo irá ficar disponível por 120 segundos. Você pode definir uma duração diferente pelo acréscimo do*intent* extra EXTRA_DISCOVERABLE_DURATION. A duração máxima que um aplicativo pode definir é 3600 segundos, e o valor 0 significa que o dispositivo estará sempre disponível. Qualquer valor abaixo de 0 ou acima de 3600 irá ser automaticamente redefinido para 120 segundos. Por exemplo, o trecho abaixo define a duração para 300:

```
Intent discoverableIntent = new
```

```
Intent(BluetoothAdapter.ACTION_REQUEST_DISCOVERABLE);

discoverableIntent.putExtra(BluetoothAdapter.EXTRA_DISCOVERABLE_DURATI
ON, 300);

startActivity(discoverableIntent);
```

Uma caixa de diálogo será exibida, pedindo permissão ao usuário para torna o dispositivo disponível, como mostrado na figura abaixo. Se o usuário responder “Yes”, então o dispositivo irá ficar disponível pela quantidade especificada de tempo. Sua Activity irá receber uma chamada de onActivityResult()), com o código do resultado sendo a duração em que o dispositivo ficará disponível. Se o usuário responder “No” ou se um erro ocorrer, o código do resultado será RESULT_CANCELED.
O dispositivo permanecerá silenciosamente nesse modo pelo tempo definido. Se você quiser ser notificado quando esse modo for alterado, pode registrar um *BroadcastReceiver* para
o*intent* ACTION_SCAN_MODE_CHANGED. Esse *intent* conterá os campos
extras EXTRA_SCAN_MODE e EXTRA_PREVIOUS_SCAN_MODE, que informarão a você o estado anterior e o atual, respectivamente. Valores possíveis para cada um deles
são: SCAN_MODE_CONNECTABLE_DISCOVERABLE, SCAN_MODE_CONNECTABLE, ou SCAN_MODE_NONE, que indicam, respectivamente, que o dispositivo está disponível, indisponível mas capaz de receber conexões, ou indisponível e incapaz de receber conexões.
Você não precisa ativar o modo de disponibilidade se for iniciar uma conexão a um servidor remoto. Ativar esse modo só é necessário quando você quiser que a sua aplicação hospede um servidor que irá aceitar

requisições de conexões, porque os dispositivos precisam ser capazes de localiza-lo antes de iniciar a conexão.

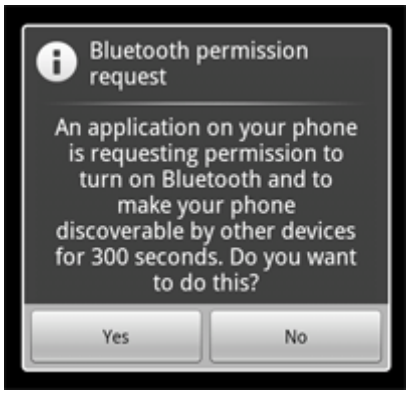


Figure 2: The enabling discoverability dialog.

Conectando os dispositivos

Para criar uma conexão entre sua aplicação e um dispositivo remoto, você precisa implementar tanto mecanismos do lado do servidor quando do lado cliente. porque um dispositivo deve abrir um servidor e o outro precisa iniciar a conexão (usando o endereço MAC do servidor). O servidor e o cliente são considerados conectados um ao outro quando cada um possuírem um BluetoothSocket conectado no mesmo canal RFCOMM. Nesse ponto, cada dispositivo pode obter *streams* de entrada pu saída e a transferência de dados pode iniciar, o que é discutido na seção “Gerenciando uma conexão”. Essa seção descreve como iniciar a conexão entre os dois dispositivos.
O servidor quanto o cliente obtém um necessário BluetoothSocket de formas diferentes. O servidor irá recebe-lo quando uma conexão é aceita. O cliente receberá-la quando for aberto um canal RFCOMM ao servidor.

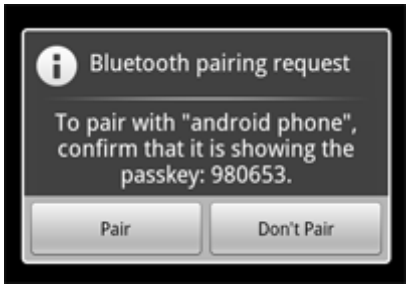


Figure 3: The Bluetooth pairing dialog.

Uma técnica de implementação é automaticamente preparar cada dispositivo como um servidor, de forma que cada um tenha um socket aberto e escutando por conexões. Dessa forma um dispositivo puder iniciar uma conexão com outro e torna-se um cliente. Alternativamente, um dispositivo pode explicitamente “hospedar” a conexão e abrir o socket sob demanda e o outro dispositivo pode simplesmente iniciar a conexão.

Nota: Se os dois dispositivo não foram emparelhados anteriormente, o Android irá automaticamente exibir uma notificação para requisitar o emparelhamento durante o processo de conexã, como mostrado na figura 3. Assim, ao tentar conectar os dispositivos, sua aplicação não precisa se preocupar com esse procedimento. Sua tentativa de conexão RFCOMM será bloqueada até que o emparelhamento seja efetuado, ou falhará se o usuário rejeitar o emparelhamento, ou se o emparelhamento demorar.

Conectando-se como um servidor

Quando você quer conectar dois dispositivos, um deles precisa agir como um servidor e abrir um BluetoothServerSocket. O propósito desse tipo de socket é ficar escutando por requisições de conexão e quando uma é aceita, fornecer um BluetoothSocket. Quando BluetoothSocket é

adquirido a partir de um BluetoothServerSocket,
o BluetoothServerSocket pode (e deve) ser descartado, a menos que você queira aceitar mas conexões.

Sobre UUID

Um *Universally Unique Identifier* (**UUID**) é um formato padronizado para ID string de 128 bits usado para identificar de forma única informação. A vantagem do UUID é que é grande o bastante para que você possa selecionar qualquer aleatoriedade e não irá ocorrer colisões. Nesse caso, é usado para identificar de forma única o serviço Bluetooth de sua aplicação. Para obter um UUID para uso em sua aplicação, você pode usar um dos muitos geradores de UUID que existem na web, e em seguida inicializar um UUID com fromString(String).
Abaixo segue um procedimento básico para configurar um socket de servidor e aceitar uma conexão:

1. Obtenha um BluetoothServerSocket através de uma chamada a listenUsingRfcommWithServiceRecord(String, UUID).A string é um nome que identifique o serviço, que o sistema irá automaticamente escrever como uma nova entrada do SDP (Service Discovery Protocol) do dispositivo (o nome é arbitrário e pode simplesmente ser o nome de sua aplicação). O UUID também é incluído junto na entrada SDP e será a base para a aceitação da conexão com o cliente. Isso é, quando o cliente tenta se conectar com o dispositivo, ele carregará um UUID que identifica unicamente o serviço com o qual deseja se conectar. Esses UUIDs precisam coincidir para que a conexão seja aceita (ver o próximo passo).

2. Inicia a espera por requisições de conexão através da accept().Esta é uma chamada bloqueante. Ela retornará algo tanto se uma conexão for aceita ou uma exceção ocorrer. Uma conexão é aceita apenas quando o dispositivo remoto envia uma solicitação de conexão com um UUID que coincida com oque está registrado nesse socket disponível. Quando for bem sucedido,accept() irá retornar um BluetoothSocket conectado.
3. A menos que você queira aceitar conexões adicionais, chame o método close().Isso irá liberar o socket do servidor e todos os recursos, mas não encerra o BluetoothSocket que está conectado, que foi retornado por accept(). Ao contrário do TCP/IP, o RFCOMM permite apenas um cliente conetado por canal ao mesmo tempo, assim na maioria dos casos faz sentido chamar o método close() para BluetoothServerSocket imediatamente após aceitar uma conexão ao socket.
   A chamada a accept() não deve ser executada na Activity principal da Interface com usuário, por ela é bloqueante e evitará outras interações com a aplicação. Normalmente faz sentido executar todo o trabalho com um BluetoothServerSocket ou BluetoothSocket em uma nova*thread* gerenciada por sua aplicação. Para encerrar o bloqueio do accept(), chame o método close() do BluetoothServerSocket (ou BluetoothSocket) a partir de uma outra *thread* e o método que efetuou o bloqueio irá ser encerrado imediatamente. Observe que todos os métodos de BluetoothServerSocket ou BluetoothSocket tomam precauções para *threads*.

Exemplo

Abaixo segue uma *thread* simplificada para o componente do servidor que aceita conexões:

```
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) { }
    }
}
```

Nesse exemplo, apenas uma conexão é desejada, assim assim que uma conexão é aceita e o BluetoothSocket é adquirido, a aplicação envia o BluetoothSocket adquirido para uma *thread* separada, encerra o BluetoothServerSocket e sai do loop.
Observe que quando accept() retorna o BluetoothSocket, o socket já está conectado, por isso você não deve chamar o método connect() (como faz do lado do cliente).
`manageConnectedSocket()` é um método ficional da aplicação que inicia a *thread* para transferência de dados, que é discutida na seção "Gerenciando uma conexão".
Você normalmente deve encerrar o seu BluetoothServerSocket assim que tiver terminado a escuta por conexões. Nesse exemplo, close() é chamado assim que um BluetoothSocket é adquirido. Você pode também desejar fornecer um método público em sua *thread* que pode encerrar o BluetoothSocket privado em caso de algum evento onde seja preciso parar de escutar no socket do servidor.

Conectando-se como um cliente

Para poder iniciar uma conexão com um dispositivo remoto (um dispositivo que possui uma socket de servidor aberto), você precisa primeiro obter um objeto BluetoothDevice que represente o dispositivo remoto. (A obtenção de um BluetoothDevice é coberto na seção acima "Descobrindo dispositivos"). Você deve então usar o BluetoothDevice para obter um BluetoothSocket e iniciar a conexão. Abaixo segue o procedimento básico:

1. Usar BluetoothDevice, obter um BluetoothSocket através da chamada ao método createRfcommSocketToServiceRecord(UUID).Isso inicializa

um BluetoothSocket que irá se conectar ao BluetoothDevice. O UUID fornecido aqui precisa coincidir com o UUID usado pelo servidor quando ele abre
seu BluetoothServerSocket (com listenUsingRfcommWithServiceRecord(String, UUID)). O uso do mesmo UUID é simplesmente uma questão de codificar a UUID no código de sua aplicação e depois referenciar ele tanto no código do servidor quanto no código do cliente.

2. Iniciar a conexão através da chamada ao método connect().Com essa chamada, o sistema executará uma busca SDP no dispositivo remoto para poder coincidir o UUID. Se a busca for bem sucedida e o dispositivo remoto aceitar a conexão, ele irá compartilhar o seu canal RFCOMM a ser usado durante a conexão e o método connect() irá retornar. Esse método é bloqueante. Se, por qualquer razão, a conexão falhar ou o método connect() demorar mais de 12 segundo, então uma exceção será disparada. Por connect() ser bloqueante, esse procedimento de conexão deve ser executado em uma *thread* separada da Activity principal.

Exemplo

Abaixo segue um exemplo de uma *thread* que inicia uma conexão Bluetooth:

```
private class ConnectThread extends Thread {

    private final BluetoothSocket mmSocket;

    private final BluetoothDevice mmDevice;



    public ConnectThread(BluetoothDevice device) {

        // Use a temporary object that is later assigned to mmSocket,
```

```
        // because mmSocket is final
        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) { }
    }
}
```

Observe que o método cancelDiscovery() é chamado antes que a conexão é feita. Você deve sempre fazer isso antes da conexão e é seguro chama-lo sem precisar verificar se há um processo de descoberta de dispositivos ativa ou não (mas se você quiser verificar isso, pode usar isDiscovering()).

`manageConnectedSocket()` é um método ficional da aplicação que inicia a *thread* para transferência de dados, discutida na próxima seção.

Quando você terminar de usar o seu BluetoothSocket, sempre chame close() para limpeza. Fazendo isso você imediatamente fechará o socket e liberará todos os recursos internos utilizados.

Gerenciando uma conexão

Após ter conectado com sucesso dois (ou mais) dispositivos, cada um terá um BluetoothSocket conectado. Aqui é onde a diversão começa porque você pode compartilhar dados entre os dispositivos.

Usando BluetoothSocket, o procedimento geral para transferência arbitrária de dados é simples:

1. Obtenha um InputStream e um OutputStream que possa manipular transmissões através do socket,
   via getInputStream() e getOutputStream(), respectivamente.
2. Leia e escreva os dados com read(byte[]) e write(byte[]).
   E é isso!

Existem, naturalmente, detalhes de implementação a considerar. Em primeiro lugar, você deve usar uma *thread* dedicada para todos os streams de leitura e escrita. Isso é importante porque tanto o método read(byte[]) quanto o write(byte[]) são métodos bloqueantes. read(byte[]) bloqueará até que haja algo para ler do stream. write(byte[]) normalmente não bloqueia nada, mas pode fazê-lo para controle de fluxo caso o dispositivo remoto não esteja chamando read(byte[]) rápido o suficiente e os buffers estiverem cheios.

Assim, seu loop principal da *thread* deve ser dedicado a leitura de dados a partir do InputStream. Um método público separado na *thread* pode ser usado para iniciar a escrita no OutputStream.

Exemplo

Abaixo segue um exemplo de como isso deve ficar:

```
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)
                        .sendToTarget();
            } catch (IOException e) {
                break;
            }
        }
    }
```

```
    /* 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) { }

    }

}
```

O construtor adquire os streams necessários e uma vez executada, a *thread* esperará por dados vindos do InputStream.
Quando read(byte[]) retorna com bytes do stream, os dados são enviados para a Activity principal usando um manipulador da classe pai. Depois volta a esperar por mais dados vindos do stream.
Enviar dados é tão simples quanto chamar o método `write()` da*thread* a partir da Activity principal passando os

bytes a serem enviados. Esse método simplesmente chama write(byte[]) para enviar os dados para o dispositivo remoto.
O método `cancel()` da *thread* é importante pois assim a conexão pode ser terminada a qualquer momento através do encerramentos do BluetoothSocket. Esse método deve sempre ser chamado quando você tiver terminado de usar a conexão Bluetooth.
Para uma demonstração do uso das APIs Bluetooth, veja Bluetooth Chat sample app.