Android 实现了对Headset 和Handsfree 两种profile 的支持。其实现核心是BluetoothHeadsetService，在PhoneApp 创建的时候会启动它。

```
 if (getSystemService(Context.BLUETOOTH_SERVICE) != null) {
  mBtHandsfree = new BluetoothHandsfree(this, phone);
  startService(new Intent(this, BluetoothHeadsetService.class));
 } else {
  // Device is not bluetooth capable
  mBtHandsfree = null;
 }
```

BluetoothHeadsetService 通过接收ENABLED_ACTION、BONDING_CREATED_ACTION 、DISABLED_ACTION 和REMOTE_DEVICE_DISCONNECT_REQUESTEDACTION 来改变状态，它也会监听Phone 的状态变化。

```
IntentFilter filter = new IntentFilter(BluetoothIntent.BONDING_CREATED_ACTION);
filter.addAction(BluetoothIntent.REMOTE_DEVICE_DISCONNECT_REQUESTED_ACTION);
filter.addAction(BluetoothIntent.ENABLED_ACTION);
filter.addAction(BluetoothIntent.DISABLED_ACTION);
registerReceiver(mBluetoothIntentReceiver, filter);
mPhone.registerForPhoneStateChanged(mStateChangeHandler,PHONE_STATE_CHANGED, null);
```

BluetoothHeadsetService 收到ENABLED_ACTION时，会先向BlueZ注册Headset 和Handsfree 两种profile（通过执行sdptool 来实现的，均作为Audio Gateway），然后让BluetoothAudioGateway 接收RFCOMM 连接，让BluetoothHandsfree 接收SCO连接（这些操作都是为了让蓝牙耳机能主动连上Android）。

```
 if (action.equals(BluetoothIntent.ENABLED_ACTION)) {
 // SDP server may not be ready, so wait 3 seconds before
 // registering records.
 // TODO: Use a different mechanism to register SDP records,
 // that actually ACK's on success, so that we can retry rather
 // than hardcoding a 3 second guess.
```

```
  mHandler.sendMessageDelayed(mHandler.obtainMessage(REGISTER_SDP_RECORDS),3000);
  mAg.start(mIncomingConnectionHandler);
  mBtHandsfree.onBluetoothEnabled();
 }
```

BluetoothHeadsetService 收到DISABLED_ACTION 时，会停止BluetoothAudioGateway 和 BluetoothHandsfree。

```
 if (action.equals(BluetoothIntent.DISABLED_ACTION)) {
  mBtHandsfree.onBluetoothDisabled();
  mAg.stop();
 }
```

Android 跟蓝牙耳机建立连接有两种方式。

1. Android 主动跟蓝牙耳机连BluetoothSettings 中和蓝牙耳机配对上之后， BluetoothHeadsetService 会收到BONDING_CREATED_ACTION，这个时候BluetoothHeadsetService 会主动去和蓝牙耳机建立RFCOMM 连接。

```
 if (action.equals(BluetoothIntent.BONDING_CREATED_ACTION)) {
  if (mState == BluetoothHeadset.STATE_DISCONNECTED) {
  // Lets try and initiate an RFCOMM connection
   try {
    mBinder.connectHeadset(address, null);
   } catch (RemoteException e) {}
  }
 }
```

RFCOMM 连接的真正实现是在ConnectionThread 中，它分两步，第一步先通过SDPClient 查询蓝牙设备时候支持Headset 和Handsfree profile。

```
 // 1) SDP query
```

```
SDPClient client = SDPClient.getSDPClient(address);
if (DBG) log("Connecting to SDP server (" + address + ")...");
if (!client.connectSDPAsync()) {
 Log.e(TAG, "Failed to start SDP connection to " + address);
 mConnectingStatusHandler.obtainMessage(SDP_ERROR).sendToTarget();
 client.disconnectSDP();
 return;
}
if (isInterrupted()) {
 client.disconnectSDP();
 return;
}

if (!client.waitForSDPAsyncConnect(20000)) { // 20 secs
 if (DBG) log("Failed to make SDP connection to " + address);
 mConnectingStatusHandler.obtainMessage(SDP_ERROR).sendToTarget();
 client.disconnectSDP();
 return;
}

if (DBG) log("SDP server connected (" + address + ")");
int headsetChannel = client.isHeadset();
if (DBG) log("headset channel = " + headsetChannel);
int handsfreeChannel = client.isHandsfree();
if (DBG) log("handsfree channel = " + handsfreeChannel);
client.disconnectSDP();
```

第2步才是去真正建立RFCOMM 连接。

```
// 2) RFCOMM connect
```

```
mHeadset = new HeadsetBase(mBluetooth, address, channel);
if (isInterrupted()) {
 return;
}
int result = mHeadset.waitForAsyncConnect(20000, // 20 secs
mConnectedStatusHandler);
if (DBG) log("Headset RFCOMM connection attempt took " +(System.currentTimeMillis() -
timestamp) + " ms");
if (isInterrupted()) {
 return;
}
if (result < 0) {
 Log.e(TAG, "mHeadset.waitForAsyncConnect() error: " + result);
 mConnectingStatusHandler.obtainMessage(RFCOMM_ERROR).sendToTarget();
 return;
} else if (result == 0) {
 Log.e(TAG, "mHeadset.waitForAsyncConnect() error: " + result +"(timeout)");
 mConnectingStatusHandler.obtainMessage(RFCOMM_ERROR).sendToTarget();
 return;
} else {
 if (DBG) log("mHeadset.waitForAsyncConnect() success");
 mConnectingStatusHandler.obtainMessage(RFCOMM_CONNECTED).sendToTarget();
}
```

当RFCOMM连接成功建立后，BluetoothHeadsetDevice 会收到RFCOMM_CONNECTED消息，它会调用 BluetoothHandsfree 来建立SCO 连接，广播通知Headset状态变化的Intent (PhoneApp 和BluetoothSettings 会接收这个Intent) 。

```
case RFCOMM_CONNECTED:
// success
```

```
if (DBG) log(" Rfcomm connected ");
if (mConnectThread != null) {
 try {
  mConnectThread.join();
 } catch (InterruptedException e) {
  Log.w(TAG, "Connect attempt cancelled, ignoring
  RFCOMM_CONNECTED", e);
  return;
 }
 mConnectThread = null;
}
setState(BluetoothHeadset.STATE_CONNECTED,BluetoothHeadset.RESULT_SUCCESS);
mBtHandsfree.connectHeadset(mHeadset, mHeadsetType);
break;
```

BluetoothHandsfree 会先做一些初始化工作，比如根据是Headset 还是Handsfree 初始化不同的ATParser，并且启动一个接收线程从已建立的RFCOMM上接收蓝牙耳机过来的控制命令（也就是AT 命令），接着判断如果是在打电话过程中，才去建立SCO 连接来打通数据通道。

```
/* package */
void connectHeadset(HeadsetBase headset, int headsetType) {
 mHeadset = headset;
 mHeadsetType = headsetType;
 if (mHeadsetType == TYPE_HEADSET) {
  initializeHeadsetAtParser();
```

```
  } else {
    initializeHandsfreeAtParser();
  }
  headset.startEventThread();
  configAudioParameters();
  if (inDebug()) {
    startDebug();
  }
  if (isIncallAudio()) {
    audioOn();
  }
 }
```

建立SCO 连接是通过SCOSocket 实现的

```
 /** Request to establish SCO (audio) connection to bluetooth
 * headset/handsfree, if one is connected. Does not block.
 * Returns false if the user has requested audio off, or if there
 * is some other immediate problem that will prevent BT audio.
 */
 /* package */
 synchronized boolean audioOn() {
  mOutgoingSco = createScoSocket();
  if (!mOutgoingSco.connect(mHeadset.getAddress())) {
    mOutgoingSco = null;
  }
  return true;
```

```
}
```

当SCO 连接成功建立后，BluetoothHandsfree 会收到SCO_CONNECTED 消息，它就会去调用AudioManager 的setBluetoothScoOn函数，从而通知音频系统有个蓝牙耳机可用了。

到此，Android 完成了和蓝牙耳机的全部连接。

```
case SCO_CONNECTED:
if (msg.arg1 == ScoSocket.STATE_CONNECTED && isHeadsetConnected()&&mConnectedSco == null) {
 if (DBG) log(" Routing audio for outgoing SCO conection ");
 mConnectedSco = (ScoSocket)msg.obj;
 mAudioManager.setBluetoothScoOn(true);
} else if (msg.arg1 == ScoSocket.STATE_CONNECTED) {
 if (DBG) log(" Rejecting new connected outgoing SCO socket ");
 ((ScoSocket)msg.obj).close();
 mOutgoingSco.close();
}
mOutgoingSco = null;
break;
```

2. 蓝牙耳机主动跟Android 连首先BluetoothAudioGateway 会在一个线程中收到来自蓝牙耳机的RFCOMM 连接，然后发送消息给BluetoothHeadsetService。

```
mConnectingHeadsetRfcommChannel = -1;
mConnectingHandsfreeRfcommChannel = -1;
if(waitForHandsfreeConnectNative(SELECT_WAIT_TIMEOUT) == false) {
 if (mTimeoutRemainingMs > 0) {
```

```
    try {
     Log.i(tag, “select thread timed out, but ” +
     mTimeoutRemainingMs + “ms of
     waiting remain.”);
     Thread.sleep(mTimeoutRemainingMs);
    } catch (InterruptedException e) {
     Log.i(tag, “select thread was interrupted (2),
     exiting”);
     mInterrupted = true;
    }
   }
  }
```

BluetoothHeadsetService 会根据当前的状态来处理消息，分3 种情况，第一是当前状态是非连接状态，会发送RFCOMM_CONNECTED 消息，后续处理请参见前面的分析。

```
 case BluetoothHeadset.STATE_DISCONNECTED:
 // headset connecting us, lets join
 setState(BluetoothHeadset.STATE_CONNECTING);
 mHeadsetAddress = info.mAddress;
 mHeadset = new HeadsetBase(mBluetooth,
mHeadsetAddress,info.mSocketFd,info.mRfcommChan,mConnectedStatusHandler);
 mHeadsetType = type;
 mConnectingStatusHandler.obtainMessage(RFCOMM_CONNECTED).sendToTarget();
 break;
```

如果当前是正在连接状态， 则先停掉已经存在的ConnectThread，并直接调用BluetoothHandsfree 去建立SCO连接。

```
 case BluetoothHeadset.STATE_CONNECTING:
 // If we are here, we are in danger of a race condition
 // incoming rfcomm connection, but we are also attempting an
```

```
// outgoing connection. Lets try and interrupt the outgoing
// connection.
mConnectThread.interrupt();
// Now continue with new connection, including calling callback
mHeadset = new
HeadsetBase(mBluetooth,mHeadsetAddress,info.mSocketFd,info.mRfcommChan,mConnectedStatusHandler);
mHeadsetType = type;
setState(BluetoothHeadset.STATE_CONNECTED,BluetoothHeadset.RESULT_SUCCESS);
mBtHandsfree.connectHeadset(mHeadset,mHeadsetType);
// Make sure that old outgoing connect thread is dead.
break;
```

如果当前是已连接的状态，这种情况是一种错误case，所以直接断掉所有连接。

```
case BluetoothHeadset.STATE_CONNECTED:
if (DBG) log(”Already connected to ” + mHeadsetAddress + “,disconnecting” +info.mAddress);
mBluetooth.disconnectRemoteDeviceAcl(info.mAddress);
break;
```

蓝牙耳机也可能会主动发起SCO 连接， BluetoothHandsfree 会接收到一个SCO_ACCEPTED消息，它会去调用AudioManager 的setBluetoothScoOn 函数，从而通知音频系统有个蓝牙耳机可用了。到此，蓝牙耳机完成了和Android 的全部连接。

```
case SCO_ACCEPTED:
if (msg.arg1 == ScoSocket.STATE_CONNECTED) {
 if (isHeadsetConnected() && mAudioPossible && mConnectedSco ==null) {
  Log.i(TAG, “Routing audio for incoming SCO connection”);
  mConnectedSco = (ScoSocket)msg.obj;
  mAudioManager.setBluetoothScoOn(true);
```

```
 } else {
   Log.i(TAG, “Rejecting incoming SCO connection”);
   ((ScoSocket)msg.obj).close();
 }
} // else error trying to accept, try again
mIncomingSco = createScoSocket();
mIncomingSco.accept();
break;
```