

SH

hi i want to understand below log from android bt system



Can you please provide the log so I can better assist you in understanding it? Without the log, it is difficult to give specific advice.



SH

app logs:-

```
02-08 12:45:28.533 21337 21337 E MainActivity: BTConnect onCreate:
02-08 12:45:28.557 21337 21337 D MainActivity: isNotiPermissionAllowed
myPackageName: com.lge.lgcompanion
02-08 12:45:28.622 21337 21337 D MainActivity: onPause: called.
02-08 12:45:50.295 21337 21337 D MainActivity: enableDisableBT: enabling BT.
02-08 12:45:50.336 21337 21337 D MainActivity: onPause: called.
```

framework logs:-

```
02-08 12:45:53.493 6057 6141 I bt_btif_vendor: bredrstartup //what is this ?
02-08 12:45:53.580 6057 6057 I bt_btif : get_profile_interface: id = handsfree //what
is this ?
02-08 12:45:53.580 6057 6057 I bt_stack: [INFO:btif_hf.cc(2641)] GetInterface
//what is this ?

02-08 12:45:53.580 6057 6155 I bt_btif_storage: btif_storage_get_adapter_property
service_mask:0x40//what is this ?

02-08 12:45:53.580 6057 6360 I bt_stack: [INFO:port_api.cc(123)]
RFCOMM_CreateConnection BDA: ff:ff:ff:ff:ff:ff

02-08 12:45:53.581 6057 6155 D bt_btif : btif_hf_upstreams_evt:
event=BTA_AG_ENABLE_EVT//what is this ?
```

There was an error generating a response

Regenerate response

Generate PNG

Download PDF

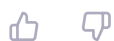
Share Link

- "MainActivity" is an activity class in your app.
- "onCreate" is a method called when the activity is created.
- "isNotiPermissionAllowed" is a method to check if the notification permission is allowed for the app.
- "enableDisableBT" is a method to enable/disable Bluetooth.
- "onpause" is a method called when the activity goes into the background.

In the framework logs:

- "bredrstartup" and "btif_hf.cc" are part of the Bluetooth HAL (Hardware Abstraction Layer) implementation.
- "btif_storage_get_adapter_property" is related to accessing adapter properties in the Bluetooth storage.
- "RFCOMM_CreateConnection" is a method related to creating a connection over the RFCOMM protocol.
- "BTA_AG_ENABLE_EVT" and "BTA_AG_REGISTER_EVT" are events related to the Bluetooth audio gateway profile.

Note: These logs give you some insight into the internal workings of the Bluetooth system on Android, but the exact meaning of each log can vary depending on the specific implementation.



SH

ok explain more about below ,what they are ?when they are used?what is thier role?
In the framework logs:

"bredrstartup" and "btif_hf.cc" are part of the Bluetooth HAL (Hardware Abstraction Layer) implementation.

"btif_storage_get_adapter_property" is related to accessing adapter properties in the Bluetooth storage.

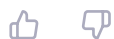
"RFCOMM_CreateConnection" is a method related to creating a connection over the RFCOMM protocol.

"BTA_AG_ENABLE_EVT" and "BTA_AG_REGISTER_EVT" are events related to the Bluetooth audio gateway profile.



1. "bredrstartup" and "btif_hf.cc": These logs are related to the Bluetooth Hardware

- Abstraction Layer (HAL) implementation. The Bluetooth HAL is an interface between the higher-level software stack and the hardware of the Bluetooth controller. The "btif_hf.cc" file is a part of the Bluetooth Hands-Free (HF) profile implementation in the Bluetooth HAL, and "bredrstartup" is likely a function call to start up the Bluetooth Basic Rate/Enhanced Data Rate (BR/EDR) radio.
2. "btif_storage_get_adapter_property": This log is related to accessing properties of the Bluetooth adapter in the Bluetooth storage. The Bluetooth adapter is the physical component that implements Bluetooth on a device, and the Bluetooth storage holds information such as paired devices, trusted devices, and device settings. The "btif_storage_get_adapter_property" function is used to retrieve a specific property of the Bluetooth adapter from the Bluetooth storage.
 3. "RFCOMM_CreateConnection": This log is related to creating a Bluetooth connection over the RFCOMM protocol. RFCOMM is a serial communication protocol used in Bluetooth for emulating RS-232 serial ports. This log message indicates that a new RFCOMM connection is being created between two Bluetooth devices.
 4. "BTA_AG_ENABLE_EVT" and "BTA_AG_REGISTER_EVT": These logs are related to the Bluetooth audio gateway (AG) profile. The AG profile is used to connect Bluetooth hands-free (HF) devices to other Bluetooth devices, such as smartphones. The "BTA_AG_ENABLE_EVT" log indicates that the AG profile is being enabled on the device, and the "BTA_AG_REGISTER_EVT" log indicates that the AG profile is being registered. These logs are related to the AG profile being started and registered so that it can be used to connect to other Bluetooth devices.



SH

ok next below logs from framework

```
02-08 12:45:53.689 21337 21337 D MainActivity: mBroadcastReceiver1: STATE  
TURNING ON
```

```
02-08 12:45:53.761 6057 6057 I bt_btif : get_profile_interface: id = avrcp
```

```
02-08 12:45:53.855 6057 6057 I bt_btif : get_profile_interface: id = a2dp
```

```
02-08 12:45:53.856 6057 6057 I btif_av : isBATEnabled: 0
```

```
02-08 12:45:53.856 6057 6057 E bt_btif : bta_av_co_get_active_peer: peer index out  
of bounds: 5
```

```
02-08 12:45:53.856 6057 6057 E bt_btif : btif_av_is_split_a2dp_enabled: No active  
peer codec config found, by default splitmode
```

```
02-08 12:45:53.856 6057 21531 I bt_osi_thread: run_thread: thread id 21531, thread  
name media_worker started
```

```
02-08 12:45:53.858 6057 6057 D bt_btif : btif_dispatch_sm_event: event: 36, len: 6
02-08 12:45:53.858 6057 6057 D bt_btif : btif_dispatch_sm_event: event 36 sent
02-08 12:45:53.858 6057 6057 D bt_btif : btif_av_state_idle_handler:
event:BTIF_SM_ENTER_EVT flags: 0 on index: 0, codec_cfg_change: 0
02-08 12:45:53.858 6057 6155 D bt_btif : ignore updating
BTIF_AV_SOURCE_CONFIG_UPDATED_EVT
```



In the framework logs:

"mBroadcastReceiver1: STATE TURNING ON" is a log statement from the MainActivity indicating that the broadcast receiver is receiving a state change event, specifically the state of Bluetooth is turning on.

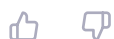
"btif_av" is part of the Bluetooth implementation and is related to the audio/video remote control profile (AVRCP).

"a2dp" stands for Advanced Audio Distribution Profile and is used for wireless transmission of high-quality audio from a source device to one or more sink devices.

"btif_dispatch_sm_event: event: 36, len: 6" indicates that the Bluetooth implementation is dispatching a state machine event with event type 36 and a length of 6 bytes.

"BTIF_SM_ENTER_EVT" is a state machine enter event.

"BTIF_AV_SOURCE_CONFIG_UPDATED_EVT" is an event related to updating the configuration of the Bluetooth audio source.



SH

anything about below logs

```
02-08 12:45:53.856 6057 6057 I btif_av : isBATEnabled: 0
02-08 12:45:53.856 6057 6057 E bt_btif : bta_av_co_get_active_peer: peer index out
of bounds: 5
02-08 12:45:53.856 6057 6057 E bt_btif : btif_av_is_split_a2dp_enabled: No active
peer codec config found, by default splitmode
```



The logs are related to the Bluetooth Advanced Audio Distribution Profile (A2DP).

- ``btif_av : isBATEnabled: 0`` refers to checking if the Bluetooth Audio Transfer

- (BAT) is enabled. If the value is `0`, it means BAT is not enabled.
`bta_av_co_get_active_peer: peer index out of bounds: 5` refers to an error in accessing the peer index for the active peer. The error is thrown because the active peer index is greater than the maximum bound of the peer list.
- `btif_av_is_split_a2dp_enabled: No active peer codec config found, by default splitmode`` refers to checking if the split mode is enabled for A2DP. Since there is no active peer codec configuration found, it falls back to default split mode.



SH

ok explain me about below
splitmode,Bluetooth Audio Transfer,default split mode



"Split mode" refers to the mode in which audio data can be split into two separate streams for concurrent transmission over Bluetooth. This is often used to enable stereo audio playback on separate left and right channels.

Bluetooth Audio Transfer is the transfer of audio data over a Bluetooth connection between two devices. The audio data can be in the form of voice or music.

The "default split mode" likely refers to a default configuration or setting that is used when no active peer codec configuration is found. In this case, the system may automatically use split mode as the default mode for Bluetooth audio transfer.



SH

ok next check below

```
02-08 12:45:53.863 6057 6360 W bt_l2cap: L2CAP - L2CA_Register() called for PSM: 0x0019
```

```
02-08 12:45:53.863 6057 6360 W bt_l2cap: L2CAP - L2CA_Register() called for PSM: 0x0017
```

```
02-08 12:45:53.863 6057 6360 W bt_l2cap: L2CAP - L2CA_Register() called for PSM: 0x001b
```

```
02-08 12:45:53.863 6057 6360 I bt_bta_av: AVRCP version used for sdp: "avrcp15"
```

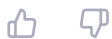


The logs contain information about the Bluetooth stack's lower layer protocols, L2CAP (Logical Link Control and Adaptation Protocol) and the Bluetooth audio/video

control protocol (AVRCP).

In the log:

- "L2CAP - L2CA_Register() called for PSM: 0x0019" indicates that the L2CAP protocol is registering with a protocol service multiplexer (PSM) value of 0x0019, which is the PSM value for the HID Control Channel in Bluetooth.
- "L2CAP - L2CA_Register() called for PSM: 0x0017" indicates that the L2CAP protocol is registering with a PSM value of 0x0017, which is the PSM value for the L2CAP Signaling Channel in Bluetooth.
- "L2CAP - L2CA_Register() called for PSM: 0x001b" indicates that the L2CAP protocol is registering with a PSM value of 0x001b, which is the PSM value for the HID Interrupt Channel in Bluetooth.
- "AVRCP version used for sdp: 'avrcp15'" indicates that the AVRCP (Audio/Video Remote Control Profile) protocol version being used for service discovery protocol (SDP) is AVRCP version 1.5.

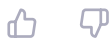


SH

what is PSM ?
why different value is assigned?



PSM stands for Protocol Service Multiplexer in Bluetooth. It is a unique identifier assigned to each Bluetooth protocol that operates on L2CAP (Logical Link Control and Adaptation Protocol) level. Different values are assigned to each PSM to differentiate between different protocols. For example, in the logs you provided, the PSM value 0x0019 is assigned to one protocol, 0x0017 to another protocol, and 0x001b to yet another protocol.



SH

nice lets continue to check below

```
02-08 12:45:53.863 6057 6360 W bt_btif : bta_av_get_rcb_by_shdl: shdl = 0,  
rcb[i].shdl = 0, rcb[i].handle = 255, rcb[i].lidx = 0
```

```
02-08 12:45:53.864 6057 6360 D bt_btif : AVRC_Open role: 1, control:67 status:0,  
handle:0
```

```
02-08 12:45:53.864 6057 6360 W bt_btif : bta_av_rc_create RC handle 0 is connected
```

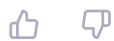
```
02-08 12:45:53.864 6057 6360 D bt_btif : rc_acp_handle: 0 idx: 1
```

```
02-08 12:45:53.864 6057 6360 D bt_btif : bta_av_rc_create create 0, role: 1, shdl:0,
rc_handle:0, lidx:6, status:0x10
02-08 12:45:53.864 6057 6155 D bt_btif : btif_av_state_idle_handler:
event:BTA_AV_REGISTER_EVT flags: 0 on index: 0, codec_cfg_change: 0
```



These logs are related to the Bluetooth audio protocol and the Bluetooth AVRCP (Audio/Video Remote Control Profile) and its connection.

- ``AVRC_Open role: 1, control:67 status:0, handle:0`` indicates that AVRCP connection is being opened in the role of the initiator (role 1) with a control channel handle of 67 and a status of success (0). The handle value of 0 may refer to the AVRCP handle.
- ``bta_av_rc_create RC handle 0 is connected`` refers to the creation of an AVRCP control block, indicating a successful connection.
- ``bta_av_rc_create create 0, role: 1, shdl:0, rc_handle:0, lidx:6, status:0x10`` shows that the AVRCP control block has been created with the role of initiator (role 1), a service handle of 0, and a control handle of 0. The lidx (list index) value of 6 and the status value of 0x10 may be used for internal tracking and error checking.



SH

i want to know how to check if register is already done


```
IntentFilter intentFilter1 = new
IntentFilter(mBluetoothAdapter.ACTION_SCAN_MODE_CHANGED);
intentFilter1.addAction(BluetoothAdapter.ACTION_STATE_CHANGED);
registerReceiver(mBroadcastReceiver2,intentFilter1);
```



To check if the ``registerReceiver`` method has already been called, you can check if the ``mBroadcastReceiver2`` receiver object has already been registered. You can do this by checking if the return value of the ``registerReceiver`` method is ``null``. If it is not ``null``, it means that the receiver has already been registered and another call to ``registerReceiver`` is not needed.

Here's an example:

java

 Copy code

```
private boolean isReceiverRegistered = false;

private void registerBluetoothReceiver() {
    if (!isReceiverRegistered) {
        IntentFilter intentFilter1 = new IntentFilter(mBluetoothAdapter.ACTION_SCAN_MO
        intentFilter1.addAction(BluetoothAdapter.ACTION_STATE_CHANGED);
        registerReceiver(mBroadcastReceiver2,intentFilter1);
        isReceiverRegistered = true;
    }
}
```



SH

ok how handle onDestroy () for the same

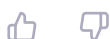


In the `onDestroy` method of your activity, you should unregister the broadcast receiver to prevent any memory leaks. You can do this by calling `unregisterReceiver` method and passing the broadcast receiver instance as a parameter, like this:

scss

 Copy code

```
@Override
protected void onDestroy() {
    super.onDestroy();
    unregisterReceiver(mBroadcastReceiver2);
}
```



SH

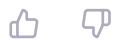
what is reciever was never registerd



"Receiver not registered" error occurs when a broadcast receiver is trying to unregister but it has not been registered previously. This may happen due to different reasons, such as:

1. The receiver was never registered in the first place.
2. The receiver was registered in a different activity or fragment.
3. The receiver was registered in a different context.
4. The receiver was unregistered before it was finished processing the broadcast.

To avoid this error, you should always make sure that a broadcast receiver is registered before attempting to unregister it. Additionally, you should ensure that the receiver is registered and unregistered in the same activity or fragment, using the same context.



SH

show me



This error occurs when you attempt to unregister a broadcast receiver that was never registered.

Here's an example:

typescript

Copy code

```
private BroadcastReceiver mBroadcastReceiver;

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    mBroadcastReceiver = new BroadcastReceiver() {
        // receiver logic here
    };
}

@Override
protected void onDestroy() {
    super.onDestroy();

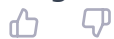
    // Unregister the receiver
    try {
```

```

        unregisterReceiver(mBroadcastReceiver);
    } catch (IllegalArgumentException e) {
        Log.e("MyApp", "Receiver was never registered", e);
    }
}

```

In this example, the receiver is only unregistered if it was previously registered. If the receiver was never registered, an `IllegalArgumentException` will be thrown and can be caught and logged for debugging purposes.



SH

How to add rssi to below function

```

private BroadcastReceiver mBroadcastReceiver3 = new BroadcastReceiver() {
    @Override
    public void onReceive(Context context, Intent intent) {
        final String action = intent.getAction();
        Log.d(TAG, "onReceive: ACTION FOUND.");

        if (action.equals(BluetoothDevice.ACTION_FOUND)) {
            BluetoothDevice device =
intent.getParcelableExtra(BluetoothDevice.EXTRA_DEVICE);
            mBTDevices.add(device);
            if (ActivityCompat.checkSelfPermission(MainActivity.this,
Manifest.permission.BLUETOOTH_CONNECT) !=
PackageManager.PERMISSION_GRANTED) {
                // TODO: Consider calling
                //   ActivityCompat#requestPermissions
                // here to request the missing permissions, and then overriding
                //   public void onRequestPermissionsResult(int requestCode, String[]
permissions,
                //                                     int[] grantResults)
                // to handle the case where the user grants the permission. See the
documentation
                // for ActivityCompat#requestPermissions for more details.
                return;
            }
            Log.d(TAG, "onReceive: " + device.getName() + ": " + device.getAddress());
            mDeviceListAdapter = new DeviceListAdapter(context,
R.layout.device_adapter_view, mBTDevices);
            lvNewDevices.setAdapter(mDeviceListAdapter);

```

```

    }
    if (action.equals(BluetoothDevice.ACTION_BOND_STATE_CHANGED)) {
        BluetoothDevice mDevice =
intent.getParcelableExtra(BluetoothDevice.EXTRA_DEVICE);
        //3 cases:
        //case1: bonded already
        if (mDevice.getBondState() == BluetoothDevice.BOND_BONDED) {
            Log.d(TAG, "BroadcastReceiver: BOND_BONDED.");
        }
        //case2: creating a bone
        if (mDevice.getBondState() == BluetoothDevice.BOND_BONDING) {
            Log.d(TAG, "BroadcastReceiver: BOND_BONDING.");
        }
        //case3: breaking a bond
        if (mDevice.getBondState() == BluetoothDevice.BOND_NONE) {
            Log.d(TAG, "BroadcastReceiver: BOND_NONE.");
        }
    }
}
};

```



To add the RSSI (received signal strength indication) to the function, you can use the `EXTRA_RSSI` extra from the `BluetoothDevice` class:

java

Copy code

```

private BroadcastReceiver mBroadcastReceiver3 = new BroadcastReceiver() {
    @Override
    public void onReceive(Context context, Intent intent) {
        final String action = intent.getAction();
        Log.d(TAG, "onReceive: ACTION FOUND.");

        if (action.equals(BluetoothDevice.ACTION_FOUND)) {
            BluetoothDevice device = intent.getParcelableExtra(BluetoothDevice.EXTRA_D
            int rssi = intent.getShortExtra(BluetoothDevice.EXTRA_RSSI, Short.MIN_VALU
            Log.d(TAG, "onReceive: " + device.getName() + ": " + device.getAddress() +
            mBTDevices.add(device);
            if (ActivityCompat.checkSelfPermission(MainActivity.this, Manifest.permis
                // TODO: Consider calling
                //     ActivityCompat#requestPermissions

```

```
// here to request the missing permissions, and then overriding
//   public void onRequestPermissionsResult(int requestCode, String[]
//                                           int[] grantResults)
// to handle the case where the user grants the permission. See the do
// for ActivityCompat#requestPermissions for more details.
return;
}
mDeviceListAdapter = new DeviceListAdapter(context, R.layout.device_adapte
lvNewDevices.setAdapter(mDeviceListAdapter);
}
if (action.equals(BluetoothDevice.ACTION_BOND_STATE_CHANGED)) {
    BluetoothDevice mDevice = intent.getParcelableExtra(BluetoothDevice.EXTRA_
    //3 cases:
    //case1: bonded already
    if (mDevice.getBondState() == BluetoothDevice.BOND_BONDED) {
        Log.d(TAG, "BroadcastReceiver: BOND_BONDED.");
    }
    //case2: creating a bond
    if (mDevice.getBondState() == BluetoothDevice.BOND_BONDING) {
        Log.d(TAG, "BroadcastReceiver: BOND_BONDING.");
    }
    //case3: breaking a bond
    if (mDevice.getBondState() == BluetoothDevice.BOND_NONE) {
        Log.d(TAG, "BroadcastReceiver: BOND_NONE.");
    }
}
}
};
```



SH

ok show me how to clear the list before showing result

