

HvacApp HvacController FrontSeatHvacController

VHAL 双向信号传递 – 核心逻辑理解 V1

deepwaterooo

May 19, 2022

Contents

1 细节版: 应用层 Controller CCC 与 BCM VHAL 双向信号传递原理: HvacController.java 理解	1
1.1 三个最重要的环节:	1
1.2 主要过程概述: 会补上	1
1.2.1 在 Controller 层面, 不管是 Daemon 还是 Client, 每个 Instance 都身背一个 mBcmSignalsCache, 用来存储当前控制器所监听的所有属性最新状态值。	1
1.3 ECU 在处理上我之前不曾有的 common sense 的机制: 每条属性可能有的有效值, idle, matching	2
1.4 自己可能弄错了的点: TimeoutRunnableTimer, 感觉这个自己完全理解得不对呀, 什么延迟发送或是延迟执行消息, 感觉全理解错了。。。。。。。。这个今天下午把它们弄懂了	2
1.5 onChangeEvent () 回调: 宏观来说, 两种情况下会收到回调	3
1.6 Incoming BCM Signals Handling: 自 BCM 向 CCC 传递的信号	3
1.7 Local Cache Handling: BCM 本地的状态临时存储图 (Map), 未必总是与硬件状态相符, 但长远看最终状态是相符的	5
1.8 Handling Broadcasting of HVAC Signals (from CCC): 来自 CCC	6
1.9 其它 HvacController 配置相关常量定义等	7
1.10 Life Cycle	8
2 (outdated, 看上面细节版就行) 关于 VHAL 将信号往上传的过程, 以 HvacController 为例	9
2.1 HvacController.java	9
3 FrontSeatHvacController: Daemon / Client Interface 不太容易理解, 多看几遍/想一想	11
3.1 关于车的来源: 每辆 Lucid 车整个系统只有一个 Car: Car vs mPropertyManager	14
3.1.1 ControllerBase.java	14
3.1.2 LucidCarService app: 在这里其实是没有任何必要的, 暂时先放这里	16
4 安卓 Handler 相关	17
4.1 mQueue 消息放入的原理总结	17
4.2 Android Handler postDelayed 的原理: 是通过阻塞实现的	17
4.2.1 如果在延迟消息还没有唤醒时又有新的消息进来, 会怎样处理呢, 看 enqueueMessage() 方法源码	18
4.3 安卓 Handler.removeMessages(int what)	19
4.4 Handler.removeMessages 的作用, 有时候为什么一定要先 remove 一下呢	19

5	HvacPresenter.java: 这里之前自己写的是, 现在的自己根本都快看不懂了呀	19
6	PresenterContract: 根据 UI 点击/滑动等事件, 向下向底层回调, 索求底层 FW/HW 硬件与 UI 同步	21
7	FrontViewFragment:	22
8	FrontSeatHvacController	23
8.1	abstract ControllerBase<T extends ControllerListenerBase> class	23
8.2	以 HvacController 为例, Controller LifeCycle 相关的理解	24
8.3	FrontSeatHvacController	24
9	BcmController	25
9.1	BcmController.BcmListener static memeber	25
9.2	Connection to VHAL	26
9.3	Incoming BCM Signals Handling	27
9.4	LucidSysUiService.java 中初始化的顺序	29
9.5	纯看代码还没有看懂, 结束打印出来的日志来看一下	30
10	VHAL TEAM 说他们现在不传这个信号给我们了, 我们需要自己来合并	31
10.1	涉及到的物理键及相关链接	31
10.2	VHAL 层面 HvacPropertyHelper.cpp 的实现逻辑	32

1 细节版: 应用层 Controller CCC 与 BCM VHAL 双向信号传递原理: HvacController.java 理解

1.1 三个最重要的环节:

- 从应用层发送请求, send/setRequest, 有没有发送成功: 这也就包括了车的联接服务有没有真正联接好, 确保发送成功了 (必要的时候插入车联接服务的 log 来确保发送成功; 要不然就有可能错怪其它组比如 VHAL 组。。。。)
- 时间上的 timeout 机制: ECU 是不是由于种种原因, 没办法成功地完成这个请求, 以致于我们收到了 timeout, 请求失败。确保我们能够收到 timeout 的日志
- onChangeEvent() 的回调: ECU 改变成功了, 那么我们就能够收到回调; 收不到回调, 在确保前两条成立, 我们发送没有出问题的话, 就可以 99% 地确信地把问题抛给 VHAL 组了。。

1.2 主要过程概述: 会补上

•

1.2.1 在 Controller 层面, 不管是 Daemon 还是 Client, 每个 Instance 都身背一个 mBcm-SignalsCache, 用来存储当前控制器所监听的所有属性最新状态值。

```
private final Map<Integer, Integer> mBcmSignalsCache = new HashMap<>();
```

- 它并不是真的存储在 BCM 层面, 而是在控制器这里当前 Controller Instance 层面。
- 主要是为了方便应用启动/重启等情况下 (应该不是应用的启动与重启, 更多的情况下应该是某个或是某些带有这些 Controller 的 Activity 或是 Fragment 的重启; Application 的重启原因不是很懂, 可能更多的时候都还是需要再次初始化的?!!!!) 初始化时快速地得到先前的状态值。

- 并不是任何时候任何情况下，只要我们需要索要某个信号，就一定得重新与车建立服务等，如果这样的话太消耗性能了，too big a overhead.....

1.3 ECU 在处理上我之前不曾有的 common sense 的机制：每条属性可能有的有效值, idle, matching

- 属性在应用层所定义的所有有效状态值，比如 true/false, 0/1, 或多个有效状态值
- Idle 状态：就是比如四个物理键之一被按下并计时了 750ms 之后，就可以认定它处于 idle 状态，不用回调通知应用层说这些键没有再被按了（之前误以为会被传回 false），而是处于 Idle 状态，不用回调???
- Match 状态：就是与上面的 Idle 比较像的另一种特殊状态，现在自己还不懂，基本只在 onChangeEvent () 的这个特殊的点会执行，应该也是一种有效的策略保护机制吧。。。。不作重点，了解即可。

```
private void handlePairedSignalsW(CarPropertyValue carPropertyValue) { // 更详细的注释在后面章节
    int bcmPropId = carPropertyValue.getPropertyId();
    int bcmValue = (Integer) carPropertyValue.getValue(); // 这个是车的硬件属性 status signal
    int cachedValue = getBcmState( bcmPropId ); // request signal
    Integer cccPropId = mMapPropIdBcm2Ccc.get( bcmPropId );
    if ( cccPropId != null && mSignalBroadcastHandler.hasMessages( cccPropId ) ) {
        // We're pending prev outbound signal ack.
        if ( bcmValue != cachedValue ) { // status signal != request signal 什么时候会出现这种情况呢？
            // BCM hasn't ack'ed receiving prev signal.
            // Drop latest updates for this signal & ignore. 这种情况下不是 新索求者胜，是新索求者弃 !!!
            Logger.d( TAG, "Pending ack of prev signal. Dropping update." );
            return;
        }
        // bcmValue == cachedValue. 也就是说, status signal == request signal
        // BCM received prev signal. Kill timeout timer. ?????
        Logger.d( TAG, "Received ack of prev signal." );
        mSignalBroadcastHandler.removeMessages( cccPropId ); // 处理完这步后，同时把 mQueue 里这一属性的所有之后的索求全部扔掉了
    }
    // No more pending ack for this signal...
    // Update local cache. 更新 BCM 层面局部存储最终的状态值：将曾经的/最终的 BCM 局部状态值，同步为硬件物理状态值，no matter what
    synchronized ( mBcmSignalsCache ) {
        mBcmSignalsCache.put( bcmPropId, bcmValue );
    }
    // Notify listener to update UI.
    notifyBcmUpdate( bcmPropId, bcmValue );

    // Start broadcasting same state to network with no timeout.
    // 就是说，这个步骤原本是可要可不要的，但是因为 ECU 的 matching 机制，还是执行这个步骤，
    // 但是 VHAL ECU 层面会有这个短路的 matching 机制，所以这里程序的逻辑不可能无限致地循环执行
    if ( cccPropId != null && mPropertyManager != null ) {
        // Special check for fan speed & temperature to only broadcast IDLE
        // when not actively sending set requests.
        int cccValue = (
            bcmPropId == IBCM_HVAC_AIRFLOW_SET_FRNT_LE || bcmPropId == IBCM_HVAC_AIRFLOW_SET_FRNT_RI
            || bcmPropId == IBCM_HVAC_AIRFLOW_SET_REAR_LE || bcmPropId == IBCM_HVAC_AIRFLOW_SET_REAR_RI
            || bcmPropId == IBCM_HVAC_TEMP_SET_FRNT_LE || bcmPropId == IBCM_HVAC_TEMP_SET_FRNT_RI
            || bcmPropId == IBCM_HVAC_TEMP_SET_REAR_LE || bcmPropId == IBCM_HVAC_TEMP_SET_REAR_RI
            ? HVAC_IDLE : bcmValue );
        // Broadcast new signal.
        Logger.d( TAG, "handlePairedSignalsW() : Broadcast new signal : cccPropId=" + cccPropId + ", cccValue=" + cccValue );
        try {
            mPropertyManager.setIntProperty( cccPropId, 0, cccValue );
        } catch ( CarNotConnectedException e ) {
            Logger.w( TAG, "broadcastHvacCmdSignalW(): Car not connected." );
        }
    }
}
```

1.4 自己可能弄错了的点: TimeoutRunnableTimer, 感觉这个自己完全理解得不对呀, 什么延迟发送或是延迟执行消息, 感觉全理解错了。。。。。。。。。。这个今天下午把它们弄懂了

1.5 onChangeEvent () 回调: 宏观来说, 两种情况下会收到回调

- 在创建 Controller Listener 的时候, 属于第一次 register callback, 会收到来自 BCM 初始值的回调
- 正常情况下, 当 CarProperty 的这个属性值真正发生 (有效, 区别于 Idle 和 match 两种机制) 状态改变的时候, 监听者 Listener 会收到回调
 - 你还记得 IVIANDLIBS-314 4/15 号的那份 log, 没有一个返回/不会收到 false 的值呢 (因为当一个键被按 750ms 之后, 键没有被按, BCM 会自动把它们当作是 idle 状态, 而不会回调传值 false)?
 - 后来这个信号传不上来的时候, 反而是在这个控制器监听都注册的时候, 会收到两个 false
 - 可是为什么这份 4/15 log 在监听者初始化的时候也从来不曾收到过 false 呢? 控制器监听者不是也还会是经历初始化的阶段吗? 还是说初始化部分的 log 被裁掉了???

```
private final BcmController.BcmListener mBcmListener = new BcmController.BcmListener() {
    @Override
    public void onEngagedHvacBtnsW(boolean bEngaged) { // 车上四个物理键被按的回调
        // 4/15 打印出来的全是 true, 后来不传这个信号回来的时候打印全是 false, 那么如果传信号, 正常初始化的时候会传 false 回来吗??
        Log.d(TAG, "onhvacbtn clicked" + bEngaged); // IcrHvacPeekWindow: onhvacbtn clickedtrue
        mBtnEngaged = bEngaged;
        if (mShowWindow)
            updateAutoCloseTimer();
        mHandler.post(mHvacBtnsRunnable); // showWindow()
    }
}
```

1.6 Incoming BCM Signals Handling: 自 BCM 向 CCC 传递的信号

```
//=====
// Incoming BCM Signals Handling: 自 BCM 向 CCC 传递的信号
//=====
private static final int HVAC_IDLE = 0xFFFF;
@Override // 监听这些属性变化: 一旦变化, 要通知应用层 UI 等
void onChangeEvent(CarPropertyValue carPropertyValue) {
    int propId = carPropertyValue.getPropertyId();
    switch (propId) {
        case IBCM_HVAC_TEMP_SET_FRNT_LE :
        case IBCM_HVAC_TEMP_SET_FRNT_RI :
        case IBCM_HVAC_TEMP_SET_REAR_LE :
        case IBCM_HVAC_TEMP_SET_REAR_RI :
            if (mHvacListener != null ) {
                int zone = (
                    propId == IBCM_HVAC_TEMP_SET_FRNT_LE ? ZONE_FRONT_LEFT :
                    propId == IBCM_HVAC_TEMP_SET_FRNT_RI ? ZONE_FRONT_RIGHT :
                    propId == IBCM_HVAC_TEMP_SET_REAR_LE ? ZONE_REAR_LEFT :
                    /* bcmPropId == IBCM_HVAC_TEMP_SET_REAR_RI ? */ ZONE_REAR_RIGHT );
                // Cap range.
                float temp = (Float)carPropertyValue.getValue();
                Logger.d( TAG, "mHvacListener.onUpdateTemperatureW(): zone=" + zone + ", temp=" + temp );
                mHvacListener.onUpdateTemperatureW( zone, temp ); // 这里已经调用了传向 UI 的监听回调
            }
            break;
    }
    if (propId == VEHL_RNG_TO_AVL) // 把这个方向稍微理解一下: 需要向应用层传递 剩余电池可走里程数的评估 信号
        handleUpdateBatteryRangeInKilometersW((Float) carPropertyValue.getValue());
    else if (propId == REAR_WINDOW_HEATING_STATUS) { // 是否开启后车窗化霜, 向 UI 传
        isRearDefrostOn = (Boolean) carPropertyValue.getValue();
        mHvacListener.onUpdateRearDefrostW(isRearDefrostOn);
        Logger.e(TAG, "mHvacListener.onUpdateRearDefrostW() is updated " + isRearDefrostOn );
    }
```

```

    } else {
        Integer cccPropId = mMapPropIdBcm2Ccc.get(propId);
        Integer bcmPropId = mMapPropIdCcc2Bcm.get(cccPropId);
        if (bcmPropId == null) { // 不成对的单向信号/单一信号????
            handleUnpairedSignalsW(carPropertyValue); // 暂时没有
        }
        // BCM 与 CCC 成对消息
    } else if (mIsSignalDaemon) { // 来自 Daemon Controller, 向下向 VHALL 向硬件处理信号
        handlePairedSignalsW(carPropertyValue); // 重中之重, 感觉理解很成困难呀.....
    } else // 如果当前 Controller 为 Daemon/Client 中的 Client Controller
        handleSignalsClientUpdateW(carPropertyValue);
    }
}

private void handleUnpairedSignalsW(CarPropertyValue carPropertyValue) {
    // No unpaired signals handling for now.
}

// 如果当前 Controller 为 Daemon/Client 中的 Client Controller
private void handleSignalsClientUpdateW(CarPropertyValue carPropertyValue) {
    int bcmPropId = carPropertyValue.getPropertyId();
    int bcmValue = (Integer) carPropertyValue.getValue();
    Logger.d( TAG, "handlePairedSignalsClientUpdateW() : bcmPropId=" + bcmPropId + ", bcmValue=" + bcmValue );
    // Update local cache.
    synchronized ( mBcmSignalsCache ) {
        mBcmSignalsCache.put( bcmPropId, bcmValue );
    }
}

// 同步 UI: 确保 UI 与硬件物理状态值相同
// Notify listener to update UI.
notifyBcmUpdate( bcmPropId, bcmValue );
}

private void handlePairedSignalsW(CarPropertyValue carPropertyValue) {
    int bcmPropId = carPropertyValue.getPropertyId();
    Log.d(TAG, "Status Prop_ID " + bcmPropId);
    int bcmValue = (Integer) carPropertyValue.getValue(); // 这个是车的硬件属性 status signal
    Log.d(TAG, "Status Prop_ID Value " + bcmValue);
    int cachedValue = getBcmState( bcmPropId ); // request signal
    Integer cccPropId = mMapPropIdBcm2Ccc.get( bcmPropId );

    Logger.d( TAG, "handlePairedSignalsW() : bcmPropId=" + bcmPropId + ", bcmValue=" + bcmValue
        + ", cachedValue=" + cachedValue + ", cccPropId=" + ( cccPropId == null ? "null" : cccPropId ) );
    // 双向信号: BCM 索求信号 mQueue 里有这个属性的 request msg, BCM-VHALL 硬件系 还没能处理完之前的索求
    if ( cccPropId != null && mSignalBroadcastHandler.hasMessages( cccPropId ) ) {
        // We're pending prev outbound signal ack.
        if ( bcmValue != cachedValue ) { // status signal != request signal 什么时候会出现这种情况呢?
            // BCM hasn't ack'ed receiving prev signal.
        }

        // BCM 已经处理完了当前这次属性更改硬件同步, 可是往 mQueue 里看看, mQueue 里还有队伍, 它说: 妈呀, 我还真是闲呀, 后面还有等着的呢!
        // 这种情况下, 仍然是 新索求者胜 !!! 等其它消息处理之后, 再反馈回去
        // 下面一句中文注释好像不对呀.....
        // to be CONFIRMED: WRONG !!! BCM 说, 你来了, (这一属性) 它们之前也提要求过, 我都还没有来得及处理呢, 哪有时间来管你? 算了吧.....
        // to be CONFIRMED: WRONG !!! BCM 说, 我还没有收到你的索求消息! 那么曾经发送的消息呢? 在 mQueue 里 (距离其消息发送 750 ms 以内; 又
        // Drop latest updates for this signal & ignore. 这种情况下不是 新索求者胜, 是新索求者弃 !!!
        Logger.d( TAG, "Pending ack of prev signal. Dropping update." );
        return;
    }
}

// 双向信号: 索求值, 与硬件物理状态值相符了; 也就是说, VHALL 硬件 终于与 BCM 索求同步了!
// 那么当前 750 ms 之内的同一属性的索求消息, 也就是 mQueue 里这一属性所有消息, 全部扔掉不管
// BCM 说, 我也是人呀, 我只能 750 ms 给你处理一条, 你不要太烦人了!!!
// bcmValue == cachedValue. 也就是说, status signal == request signal
// BCM received prev signal. Kill timeout timer. ???
Logger.d( TAG, "Received ack of prev signal." );
// mQueue 里能有的消息, 一定是 750 ms 之内的消息吗? 好像是呀
mSignalBroadcastHandler.removeMessages( cccPropId ); // 处理完这步后, 同时把 mQueue 里这一属性的所有之后的索求全部扔掉了
}

// 消息桶 mQueue 里没有这一属性的消息了: 这个属性的短期内 (750ms) 最终状态 (这里应该不存在 cccProdId == null 的情况)
// BCM 本地索求值 同步为 短期内 (750ms) 硬件物理最终状态
// No more pending ack for this signal...
// Update local cache. 更新 BCM 层面局部存储最终的状态值: 将曾经的/最终的 BCM 局部状态值, 同步为硬件物理状态值, no matter what
synchronized ( mBcmSignalsCache ) {
    mBcmSignalsCache.put( bcmPropId, bcmValue );
}

// Notify listener to update UI. 当是这条属性的当前/短期 750ms 内的最终状态, 需要反馈到应用层 UI 等
// 也因为 UI 最初同步为 request signal 的更新, 可能并不成功, 或与当前值不符, 需要 UI 无条件同步为硬件状态值 ?????
// 他说, 还有一个更重要的原因是什么来着? timeout 机制 ????? 我忘记了.....
notifyBcmUpdate( bcmPropId, bcmValue );

// 怎么这里又来了一遍呢? 这里这些不同层面的广播传递/事件回调等的执行顺序是怎样的, 昏了..... ??????????????????

```

```
// 这里是什么呢??????
// Start broadcasting same state to network with no timeout.
if ( cccPropId != null && mPropertyManager != null ) {
    // Special check for fan speed & temperature to only broadcast IDLE
    // when not actively sending set requests.
    int cccValue = (
        bcmPropId == IBCM_HVAC_AIRFLOW_SET_FRNT_LE || bcmPropId == IBCM_HVAC_AIRFLOW_SET_FRNT_RI
        || bcmPropId == IBCM_HVAC_AIRFLOW_SET_REAR_LE || bcmPropId == IBCM_HVAC_AIRFLOW_SET_REAR_RI
        || bcmPropId == IBCM_HVAC_TEMP_SET_FRNT_LE || bcmPropId == IBCM_HVAC_TEMP_SET_FRNT_RI
        || bcmPropId == IBCM_HVAC_TEMP_SET_REAR_LE || bcmPropId == IBCM_HVAC_TEMP_SET_REAR_RI
        ? HVAC_IDLE : bcmValue );
    // Broadcast new signal.
    Logger.d( TAG, "handlePairedSignalsW() : Broadcast new signal : cccPropId=" + cccPropId + ", cccValue=" + cccValue
    try { // 怎么这里又来了一遍呢？这里这些不同层面的广播传递/事件回调等的执行顺序是怎样的，昏了.....
        mPropertyManager.setIntProperty( cccPropId, 0, cccValue );
    } catch ( CarNotConnectedException e ) {
        Logger.w( TAG, "broadcastHvacCmdSignalW(): Car not connected." );
    }
}
}
```

1.7 Local Cache Handling: BCM 本地的状态临时存储图 (Map)，未必总是与硬件状态相符，但长远看最终状态是相符的

```
//=====
// Local Cache Handling: BCM 本地的状态临时存储图 (Map)，未必总是与硬件状态相符，但长远看最终状态是相符的
//=====
// BCM (VHAL 层面) 有一个它自己的当前状态存储值，未必与硬件状态值相同，必要时需要改写重置
private final Map<Integer, Integer> mBcmSignalsCache = new HashMap<>();

// 它是写在/设置在某个地方的某条性能配置属性
private static final String SYSTEM_PROPERTY_HVAC_SIGNALS_TIMEOUT = "persist.data.hvac.signals.timeout";
private int mHvacSignalsTimeoutMillis = 750; // Default Value

private int getBcmState(int bcmPropId) {
    Integer value;
    synchronized ( mBcmSignalsCache ) {
        value = mBcmSignalsCache.get( bcmPropId );
    }
    return ( value == null ? 0 : value );
}

// 那么下面的两个方法：什么时候直接发送消息，什么时候也需要设置 BCM 本地状态呢？
// setBcmState () 多两个步骤：
// 1. 第一要事：把 request Signal 存储到 BCM 层的本地临时存储 (此时，未必，或者更确切地说一定不会，与硬件 status Signal 相符)
// 2. notifyBcmUpdate( bcmPropId, bcmValue ); 也就是说，不管 BCM VHAL 硬件与索求同步没有，先将应用层 UI 同步为所索求的理想状态
// 相同点：都向下向 BCM 发送消息。可能 sendCcc2Bcm 并没有应用层界面或是 UI 需要严格要求与索求值同步之类的吧????????
private void sendCcc2Bcm(int bcmPropId, int bcmValue) { // 由 CCC 向 BCM 向下向 VHAL 向硬件发送索求消息，有些消息需要延迟执行
    Logger.d( "RearDefrost", "setBcmState(): bcmPropId=" + bcmPropId + ", bcmValue=" + bcmValue );
    // Set new state to local cache.
    mSignalBroadcastHandler.removeMessages( bcmPropId );
    mSignalBroadcastHandler.sendMessage( mSignalBroadcastHandler
        .obtainMessage( MSG_BROADCAST_SIGNAL, bcmPropId, bcmValue ) );
    // Set 750ms timeout timer.
    if ( mHvacSignalsTimeoutMillis > 0 ) // 延迟 mHvacSignalsTimeoutMillis 这么长时间，再去执行这个属性的所有 ??? 消息
        mSignalBroadcastHandler.sendMessageDelayed( bcmPropId, mHvacSignalsTimeoutMillis );
}

private void setBcmState(int bcmPropId, int bcmValue) { // 信号主要 (一定是 ???) 来自 CCC 吗？我觉得是，但是仍然感觉不够确定
    Logger.d( TAG, "setBcmState(): bcmPropId=" + bcmPropId + ", bcmValue=" + bcmValue );
    // Set new state to local cache.
    // 第一要事：把 request Signal 存储到 BCM 层的本地临时存储 (此时，未必，或者更确切地说一定不会，与硬件 status Signal 相符)
    synchronized ( mBcmSignalsCache ) {
        mBcmSignalsCache.put( bcmPropId, bcmValue );
    }

    // Notify listener to update UI. ?????? 这里不应该是来自 CCC 的信号向下向 BCM 传递才对吗????????
    // to be CONFIRMED: BCM 信号向下传递平均需要 750 ms，为了应用层 UI 与硬件属性状态尽可能地相符，在这里，
    // 向上向应用层 UI，向下向 VHAL 硬件层，同步向两个方向发送状态更改 request? 这里的理解需要确认
    notifyBcmUpdate( bcmPropId, bcmValue );
    // eg: HvacApp 说要求，左前驾 70 度，当前物理状态 65 度；这里是要求立即马上，把 UI 同步为索求值 70 度，不管 BCM, VHAL 与硬件同

    // Start broadcasting new state to network with 750ms timeout.
    // CCC 向 BCM 发送 属性状态改变索求消息：
    // 将 BCM 信号转变为 CCC 信号，通过消息机制，向下向 BCM/VHAL 硬件发消息请求 FW 硬件等与其同步，必要时设置 750 ms timeout
```

```

Integer cccPropId = mMapPropIdBcm2Ccc.get( bcmPropId );
if ( cccPropId != null ) {
    mSignalBroadcastHandler.removeMessages( cccPropId ); // 750 ms 内的竞争机制：新者胜 !!!
    mSignalBroadcastHandler.sendMessage( mSignalBroadcastHandler
        .obtainMessage( MSG_BROADCAST_SIGNAL, cccPropId, bcmValue ) );

    // Set 750ms timeout timer.
    if ( mHvacSignalsTimeoutMillis > 0 )
        // 将这个属性的那条消息（只有一条，750 内也就是 mQueue 中尚未处理的该属性消息，只剩一条），延迟 750 ms 发送
        mSignalBroadcastHandler.sendMessageDelayed( cccPropId, mHvacSignalsTimeoutMillis );
}
}

// 举一个调用 setBcmState( bcmPropId, temperature ) 的例子
public void setTemperature(int zone, int temperature) {
    if ( !mIsSignalDaemon ) {
        setTemperatureClient( zone, temperature );
        return;
    }
    Logger.d( TAG, "setTemperature() : zone=" + zone + ", temperature=" + temperature );
    int bcmPropId = (
        zone == ZONE_FRONT_LEFT ? IBCM_HVAC_TEMP_SET_FRNT_LE :
        zone == ZONE_FRONT_RIGHT ? IBCM_HVAC_TEMP_SET_FRNT_RI :
        zone == ZONE_REAR_LEFT ? IBCM_HVAC_TEMP_SET_REAR_LE :
        zone == ZONE_REAR_RIGHT ? IBCM_HVAC_TEMP_SET_REAR_RI :
        0 );
    setBcmState( bcmPropId, temperature ); // setBcmState(id, val)
}

// 通知相关属性的向上的应用层 APP 层 UI 层监听者，这个东西有过变化（属性名，属性的状态值）
private void notifyBcmUpdate(int bcmPropId, int bcmValue) { // 向上传递，向应用层向 UI 传递
    // Notify listener.
    switch ( bcmPropId ) {
        //..... 很多略过，总之就是，通知相关属性的应用层监听者，这个东西有过变化（属性名，属性的状态值）
        case IBCM_HVAC_MOD_SET_FRNT_LE :
        case IBCM_HVAC_MOD_SET_FRNT_RI :
        case IBCM_HVAC_MOD_SET_REAR_LE :
        case IBCM_HVAC_MOD_SET_REAR_RI :
            if ( mHvacListener != null ) {
                int zone = (
                    bcmPropId == IBCM_HVAC_MOD_SET_FRNT_LE ? ZONE_FRONT_LEFT :
                    bcmPropId == IBCM_HVAC_MOD_SET_FRNT_RI ? ZONE_FRONT_RIGHT :
                    bcmPropId == IBCM_HVAC_MOD_SET_REAR_LE ? ZONE_REAR_LEFT :
                    /* bcmPropId == IBCM_HVAC_MOD_SET_REAR_RI ? */ ZONE_REAR_RIGHT );
                boolean isAutoOn = ( ( bcmValue & SIG_MASK_AUTO ) != 0 ); // 自动风扇是开着的
                Log.d(TAG, "mHvacListener.onUpdateAutoW() : zone=" + zone + ", isAutoOn=" + isAutoOn +
                    " bcm value : " + bcmValue + " SIG_MASK_AUTO " + SIG_MASK_AUTO + " BCM prop ID : " + bcmPropId );
                mHvacListener.onUpdateAutoW( zone, isAutoOn );
                //..... 其它尾部相关逻辑
            }
            break;
    }
}
}

```

1.8 Handling Broadcasting of HVAC Signals (from CCC): 来自 CCC

- 这里主要是自定义了一个消息机制的 Handler，用来处理交互过程中的各种交互消息任务

```

//=====
// Handling Broadcasting of HVAC Signals (from CCC): 来自 CCC
//=====
private static final int MSG_BROADCAST_SIGNAL = 100;
private HandlerThread mHandlerThread;
private SignalBroadcastHandler mSignalBroadcastHandler;
class SignalBroadcastHandler extends Handler { //
    SignalBroadcastHandler(Looper looper) {
        super( looper );
    }
    @Override // 这里，真正处理消息的时候，车的硬件配置才与消息 request 相符吧，不是发送的时候
    public void handleMessage(Message msg) {
        if ( msg.what == MSG_BROADCAST_SIGNAL ) {
            broadcastCccSignalW( msg.arg1, msg.arg2 );
        } else // 好像这个方法永远也不会被调用呀..... ??????????????????????
            cancelBroadcastCccSignalW( msg.what );
    }
    void broadcastCccSignalW(int cccSignal, int cccValue) {
        Logger.d( TAG, "broadcastCccSignalW() : cccSignal=" + cccSignal + ", cccValue=" + cccValue );
    }
}

```

```

// Start broadcasting signal
if ( mPropertyManager == null )
    return;
try { // 当把索求 request signal 写进了 mPropertyManager, 由安卓车载系统车的软件 FW 去处理硬件与之同步相关的逻辑事情
    mPropertyManager.setIntProperty( cccSignal, 0, cccValue );
} catch ( CarNotConnectedException e ) {
    Logger.w( TAG, "broadcastHvacCmdSignalW(): Car not connected." );
}
}

void cancelBroadcastCccSignalW(int cccPropId) { // 这个, 还没有看, 要把这些都看懂
// Broadcasting timeout occurred.
Integer propId = mMapPropIdCcc2Bcm.get(cccPropId);
if (propId != REAR_WINDOW_HEATING_STATUS) {
    Logger.d(TAG, "Rear Window Heating Status PROP_ " + propId);
    int bcmPropId = (propId == null ? 0 : propId);

    // Reset local cache to what bcm was broadcasting.
    int bcmValue = 0;
    try {
        bcmValue = mPropertyManager.getIntProperty(bcmPropId, 0);
    } catch (CarNotConnectedException e) {
        Logger.w(TAG, "cancelBroadcastCccSignalW(): car not connected.");
    }

    Logger.d(TAG, "cancelBroadcastCccSignalW() : cccPropId=" + cccPropId
        + ", bcmPropId=" + bcmPropId + ", bcmValue=" + bcmValue);

    synchronized (mBcmSignalsCache) {
        mBcmSignalsCache.put(bcmPropId, bcmValue);
    }

    // Notify listener to update UI.
    notifyBcmUpdate(bcmPropId, bcmValue);

    int cccValue = bcmValue;
    // Special check for fan speed and temperature.
    // If so, resetting cccValue back to HVAC_IDLE
    if (bcmPropId == IBCM_HVAC_AIRFLOW_SET_FRNT_LE || bcmPropId == IBCM_HVAC_AIRFLOW_SET_FRNT_RI
        || bcmPropId == IBCM_HVAC_AIRFLOW_SET_REAR_LE || bcmPropId == IBCM_HVAC_AIRFLOW_SET_REAR_RI
        || bcmPropId == IBCM_HVAC_TEMP_SET_FRNT_LE || bcmPropId == IBCM_HVAC_TEMP_SET_FRNT_RI
        || bcmPropId == IBCM_HVAC_TEMP_SET_REAR_LE || bcmPropId == IBCM_HVAC_TEMP_SET_REAR_RI) {
        cccValue = HVAC_IDLE;
    }

    // Re-broadcast bcm signal.
    broadcastCccSignalW(cccPropId, cccValue);
}
}
};

```

1.9 其它 HvacController 配置相关常量定义等

```

// HvacController 中的一个同步锁
private final Object mPropertiesLock = new Object();
private static final Integer[][] S_HVAC_PROPERTIES = new Integer[][] {
    // { IBCM_HVAC_CLIMATE_MODE_SET, ICCV_HVAC_CLIMATE_MODE_FROM_HMI },
    // { IBCM_HVAC_CLIMT_CODE_SET, ICCV_HVAC_CLIMT_CODE_FRM_HMI },
    // { IBCM_HVAC_SYNC_SET, ICCV_HVAC_SYNC_FROM_HMI },
    // { IBCM_HVAC_MOD_SET_FRNT_LE, ICCV_HVAC_MOD_FROM_HMI_FRNT_LE },
    // { IBCM_HVAC_MOD_SET_FRNT_RI, ICCV_HVAC_MOD_FROM_HMI_FRNT_RI },
    // { IBCM_HVAC_MOD_SET_REAR_LE, ICCV_HVAC_MOD_FROM_HMI_REAR_LE },
    // { IBCM_HVAC_MOD_SET_REAR_RI, ICCV_HVAC_MOD_FROM_HMI_REAR_RI },
    // { IBCM_HVAC_DISTBN_SET_FRNT_LE, ICCV_HVAC_DISTBN_FROM_HMI_FRNT_LE },
    // { IBCM_HVAC_DISTBN_SET_FRNT_RI, ICCV_HVAC_DISTBN_FROM_HMI_FRNT_RI },
    // { IBCM_HVAC_DISTBN_SET_REAR_LE, ICCV_HVAC_DISTBN_FROM_HMI_REAR_LE },
    // { IBCM_HVAC_DISTBN_SET_REAR_RI, ICCV_HVAC_DISTBN_FROM_HMI_REAR_RI },
    // { IBCM_HVAC_AIRFLOW_SET_FRNT_LE, ICCV_HVAC_AIRFLOW_SET_FROM_HMI_FRNT_LE },
    // { IBCM_HVAC_AIRFLOW_SET_FRNT_RI, ICCV_HVAC_AIRFLOW_SET_FROM_HMI_FRNT_RI },
    // { IBCM_HVAC_AIRFLOW_SET_REAR_LE, ICCV_HVAC_AIRFLOW_SET_FROM_HMI_REAR_LE },
    // { IBCM_HVAC_AIRFLOW_SET_REAR_RI, ICCV_HVAC_AIRFLOW_SET_FROM_HMI_REAR_RI },
    { IBCM_HVAC_TEMP_SET_FRNT_LE, ICCV_HVAC_TEMP_SET_FROM_HMI_FRNT_LE },
    { IBCM_HVAC_TEMP_SET_FRNT_RI, ICCV_HVAC_TEMP_SET_FROM_HMI_FRNT_RI },
    { IBCM_HVAC_TEMP_SET_REAR_LE, ICCV_HVAC_TEMP_SET_FROM_HMI_REAR_LE },
    { IBCM_HVAC_TEMP_SET_REAR_RI, ICCV_HVAC_TEMP_SET_FROM_HMI_REAR_RI },
};

```

```

// { REAR_WINDOW_HEATING_STATUS, REAR_WINDOW_HEATING_REQ },
// { BCM_STEERING_HEAT_ST, ICC_STEERING_HEAT_REQ }
};

private final Map<Integer, Integer> mMapPropIdBcm2Ccc = new HashMap<>();
private final Map<Integer, Integer> mMapPropIdCcc2Bcm = new HashMap<>();
// BCM (VHAL 层面) 也有一个它自己的当前状态存储值, 未必与硬件状态值相同, 必要情况下需要改写重置
private final Map<Integer, Integer> mBcmSignalsCache = new HashMap<>();

private static final int IBCM_HVAC_TEMP_SET_FRNT_LE = 0x21601250; // IBCM_HVAC_TEMP_SET_FRNT_LE 0x21601250 previously 0x214
private static final int ICC_HVAC_TEMP_SET_FROM_HMI_FRNT_LE = 0x21601251;
private static final int IBCM_HVAC_TEMP_SET_FRNT_RI = 0x21601252; // IBCM_HVAC_TEMP_SET_FRNT_RI 0x21601252 previously 0x214
private static final int ICC_HVAC_TEMP_SET_FROM_HMI_FRNT_RI = 0x21601253;
private static final int IBCM_HVAC_TEMP_SET_REAR_LE = 0x21601254; // IBCM_HVAC_TEMP_SET_REAR_LE 0x21601254 previously 0x214
private static final int ICC_HVAC_TEMP_SET_FROM_HMI_REAR_LE = 0x21601255;
private static final int IBCM_HVAC_TEMP_SET_REAR_RI = 0x21601256; // IBCM_HVAC_TEMP_SET_REAR_RI 0x21601256 previously 0x214
private static final int ICC_HVAC_TEMP_SET_FROM_HMI_REAR_RI = 0x21601257;

private float batteryDistanceKmStatus; // 大概是说, 剩下的电池还可以走 XX 公里
// 是一个跟电池相关/导航相关的某个信号标志, 当它等于某个特定的值, 需要将某个或某些信号传向 UI 传向上层应用层
private static final int VEH_RNG_TO_AVL = 0x21601802;
private final static int[] S_NOTIFY_PROP_IDS = new int[]{
    VEH_RNG_TO_AVL
};
// // 剩下的电池, 还够走 XX 公里: 这个不重要, 暂时略去
// void handleUpdateBatteryRangeInKilometersW(Float value) {
//     synchronized (mPropertiesLock) {
//         batteryDistanceKmStatus = value;
//     }
// }
// // 应用层面的回调默认实现, 向 UI 传递剩余电池可走里程数评估, HvacApp LucidCarControls APP 中有用到
// public float getCurrentBatteryKilometers() {
//     synchronized (mPropertiesLock) {
//         return batteryDistanceKmStatus;
//     }
// }

```

1.10 Life Cycle

```

//=====
// Life Cycle
//=====

private Context mContext;
private HvacListener mHvacListener;
private boolean mIsSignalDaemon = false;

public HvacController() {
    for (Integer[] sHvacProp : S_HVAC_PROPERTIES) {
        mMapPropIdBcm2Ccc.put(sHvacProp[0], sHvacProp[1]);
        mMapPropIdCcc2Bcm.put(sHvacProp[1], sHvacProp[0]);
    }
    // Signal broadcasting worker thread.
    mHandlerThread = new HandlerThread( "SignalBroadcastHandlerThread" );
    mHandlerThread.start();
    mSignalBroadcastHandler = new SignalBroadcastHandler( mHandlerThread.getLooper() );
}

@Override
List<Integer> getStatusSignals() {
    List<Integer> statusSignals = new ArrayList<>();
    for (Integer[] sHvacProp : S_HVAC_PROPERTIES)
        statusSignals.add(sHvacProp[0]);
    for (Integer propId : S_NOTIFY_PROP_IDS) // 这个极特殊的信号, 也需要处理一下
        statusSignals.add(propId);
    return statusSignals;
}

@Override
public void init(HvacListener hvacListener) {
    mHvacListener = hvacListener;
    super.init(hvacListener);
}

@Override
public void init(Context context, HvacListener hvacListener) {
    mContext = context;
    // Using configs to override for debugging.
}

```

```
// 这是一个决定性能的非常重要的属性更改 timeout 阀门 750 ms: 写在某个配置的地方, 可以获取, 应该也可以变更
String propTimeout = SystemPropertiesProxy.get( mContext,
                                             SYSTEM_PROPERTY_HVAC_SIGNALS_TIMEOUT, "" );

// 如果有个性化配置, 那么这里可以同步一下, 更新缺省值
try { mHvacSignalsTimeoutMillis = Integer.parseInt( propTimeout ); }
catch ( NumberFormatException e ) { /* Do nothing */ }
mHvacListener = hvacListener;
super.init(context, hvacListener);
}

public void setSignalDaemon(boolean isSignalDaemon) {
    mIsSignalDaemon = isSignalDaemon;
// 如果是 Daemon, 那么作为广播信号的接收者, 就需要注册各种 IntentFilter action, 以进行过滤
    if ( mIsSignalDaemon ) {
        IntentFilter intentFilter = new IntentFilter();
        for ( String action : DAEMON_INTENT_FILTER )
            intentFilter.addAction( action );
        mContext.registerReceiver( mDaemonClientReceiver, intentFilter ); // 注册接收过滤器
    } else {
        mContext.unregisterReceiver( mDaemonClientReceiver );
    }
}
}
```

2 (outdated, 看上面细节版就行) 关于 VHAL 将信号往上传的过程, 以 HvacController 为例

2.1 HvacController.java

- 这里的逻辑怎么还是被自己读得是懂非懂的呢?

```
//=====
// Incoming BCM Signals Handling
//=====
private static final int HVAC_IDLE = 0xFFFF;
@Override
void onChangeEvent(CarPropertyValue carPropertyValue) {
    int propId = carPropertyValue.getPropertyId();
    if (propId == VEH_RNG_TO_AVL)
        handleUpdateBatteryRangeInKilometersW((Float) carPropertyValue.getValue());
    else if (propId == REAR_WINDOW_HEATING_STATUS) {
        isRearDefrostOn = (Boolean) carPropertyValue.getValue();
        mHvacListener.onUpdateRearDefrostW(isRearDefrostOn);
        Logger.e(TAG, "mHvacListener.onUpdateRearDefrostW() is updated " + isRearDefrostOn );
    } else {
        Integer cccPropId = mMapPropIdBcm2Ccc.get(propId);
        Integer bcmPropId = mMapPropIdCcc2Bcm.get(cccPropId);
        if (bcmPropId == null) { // No unpaired signals handling for now.
            handleUnpairedSignalsW(carPropertyValue);
        } else if (mIsSignalDaemon) {
            handlePairedSignalsW(carPropertyValue);
        } else {
            handleSignalsClientUpdateW(carPropertyValue);
        }
    }
}

private void handleUnpairedSignalsW(CarPropertyValue carPropertyValue) {
    // No unpaired signals handling for now.
}

private void handlePairedSignalsW(CarPropertyValue carPropertyValue) {
    int bcmPropId = carPropertyValue.getPropertyId();
    Log.d(TAG, "Status Prop_ID " + bcmPropId);
    int bcmValue = (Integer) carPropertyValue.getValue();
    Log.d(TAG, "Status Prop_ID Value " + bcmValue);
    int cachedValue = getBcmState( bcmPropId );
    Integer cccPropId = mMapPropIdBcm2Ccc.get( bcmPropId );
    Logger.d( TAG, "handlePairedSignalsW() : bcmPropId=" + bcmPropId + ", bcmValue=" + bcmValue
        + ", cachedValue=" + cachedValue + ", cccPropId=" + ( cccPropId == null ? "null" : cccPropId ) );
    if ( cccPropId != null && mSignalBroadcastHandler.hasMessages( cccPropId ) ) {
// 当两个值不等的时候, 这是一种情况
// We're pending prev outbound signal ack.
        if ( bcmValue != cachedValue ) {
```

```

        // BCM hasn't ack'ed receiving prev signal.
        // Drop latest updates for this signal & ignore.
        Logger.d( TAG, "Pending ack of prev signal. Dropping update. " );
        return;
    }
    // 当两个值相等的时候：这里我好像还是没有想明白，就是之前的与现在的，是一样的，所以不用再怎样了???
    // bcmValue == cachedValue.
    // BCM received prev signal. Kill timeout timer.
    Logger.d( TAG, "Received ack of prev signal. " );
    mSignalBroadcastHandler.removeMessages( cccPropId );
}
// No more pending ack for this signal...
// Update local cache. BCM 有一个当前缓存值，VHAL 层面也有一个当前缓存的值 ??? 把笔记本找出来再看一下。
synchronized ( mBcmSignalsCache ) {
    mBcmSignalsCache.put( bcmPropId, bcmValue );
}
// Notify listener to update UI. 什么情况是有必要再次更新 UI 的呢？
notifyBcmUpdate( bcmPropId, bcmValue );
// Start broadcasting same state to network with no timeout.
if ( cccPropId != null && mPropertyManager != null ) {
    // Special check for fan speed & temperature to only broadcast IDLE
    // when not actively sending set requests.
    int cccValue = ( // 为什么要判断这么一大堆的状态???
        bcmPropId == IBCM_HVAC_AIRFLOW_SET_FRNT_LE || bcmPropId == IBCM_HVAC_AIRFLOW_SET_FRNT_RI
        || bcmPropId == IBCM_HVAC_AIRFLOW_SET_REAR_LE || bcmPropId == IBCM_HVAC_AIRFLOW_SET_REAR_RI
        || bcmPropId == IBCM_HVAC_TEMP_SET_FRNT_LE || bcmPropId == IBCM_HVAC_TEMP_SET_FRNT_RI
        || bcmPropId == IBCM_HVAC_TEMP_SET_REAR_LE || bcmPropId == IBCM_HVAC_TEMP_SET_REAR_RI
        ? HVAC_IDLE : bcmValue );
    // Broadcast new signal.
    Logger.d( TAG, "handlePairedSignalsW() : Broadcast new signal : cccPropId=" + cccPropId + ", cccValue=" + cccValue );
    try {
        mPropertyManager.setIntProperty( cccPropId, 0, cccValue );
    } catch ( CarNotConnectedException e ) {
        Logger.w( TAG, "broadcastHvacCmdSignalW(): Car not connected. " );
    }
}
}
}

```

- 找一片日志出来对照着看一下：好是需要好好把这一个交接层的代码给弄明白了!!!!!!

```

LucidVehicleHalImpl: get_hvac_hw_switch_prop: HVAC h/w button engaged
LucidVehicleHalImpl: doHalEvent prop_val.prop = 0x21201301 for signal InfoT_Body/HVAC_Status/IHVAC_Key2Sts = 1.000000
ActivityManager: Receiver with filter android.content.IntentFilter@2f66aeb already registered for pid 4797, callerPackage i
LucidVehicleHalImpl: doHalEvent prop_val.prop = 0x2150140f for signal TCU/TCU_EPOCHTime/ITCU_Epoch_Time_Sec = 1650055501.00
LucidVehicleHalImpl: doHalEvent prop_val.prop = 0x21401244 for signal InfoT_Body/BCM_HVAC_1/IBCM_HvacAirFlowSetRearLe = 70.
LucidVehicleHalImpl: doHalEvent prop_val.prop = 0x21401246 for signal InfoT_Body/BCM_HVAC_1/IBCM_HvacAirFlowSetRearRi = 70.
LucidVehicleHalImpl: doHalEvent prop_val.prop = 0x21401242 for signal InfoT_Body/BCM_HVAC_1/IBCM_HvacAirFlowSetFrntRi = 70.
LucidVehicleHalImpl: doHalEvent prop_val.prop = 0x21401240 for signal InfoT_Body/BCM_HVAC_1/IBCM_HvacAirFlowSetFrntLe = 70.
LucidVehicleHalImpl: doHalEvent prop_val.prop = 0x21601805 for signal Powertrain/BMU_E2E/IBMU_StatsPkPwr = -2.200000
LucidVehicleHalImpl: doHalEvent prop_val.prop = 0x21611403 for signal TCU/TCU_GPS_INFO/ITCU_GPS_Latitude = 135152218.000000
chatty : uid=1001000(com.lucid.car.lucidota) SignalBroadcast expire 105 lines
HvacController: Status Prop_ID 557847108
HvacController: Status Prop_ID Value 70
HvacController: [HvacController.onChangeEvent():1258] handlePairedSignalsW(): bcmPropId=557847108, bcmValue=70, cachedValue=70
HvacController: [HvacController.access$1800():29] mHvacListener.onUpdateFanSpeedW(): zone=33, fanSpeed=70
HvacController: [HvacController.onChangeEvent():1258] handlePairedSignalsW(): Broadcast new signal : cccPropId=557847109, cccValue=70

chatty : uid=1001000(com.lucid.car.lucidota) SignalBroadcast expire 94 lines // 不知道这一行是上面还是下面的
LucidVehicleHalImpl: Received VehicleProperty: 0x21401245
LucidVehicleHalImpl: sending: InfoT_Body/CCC_HVAC_1/ICCC_HvacAirFlowSetFromHmiRearLe 65535.000000
LucidVehicleHalImpl: Received VehicleProperty: 0x21401247
LucidVehicleHalImpl: sending: InfoT_Body/CCC_HVAC_1/ICCC_HvacAirFlowSetFromHmiRearRi 65535.000000
LucidVehicleHalImpl: Received VehicleProperty: 0x21401243
LucidVehicleHalImpl: sending: InfoT_Body/CCC_HVAC_1/ICCC_HvacAirFlowSetFromHmiFrntRi 65535.000000
LucidVehicleHalImpl: Received VehicleProperty: 0x21401241
LucidVehicleHalImpl: sending: InfoT_Body/CCC_HVAC_1/ICCC_HvacAirFlowSetFromHmiFrntLe 65535.000000
LucidVehicleHalImpl: Received VehicleProperty: 0x21401245
LucidVehicleHalImpl: sending: InfoT_Body/CCC_HVAC_1/ICCC_HvacAirFlowSetFromHmiRearLe 65535.000000
LucidVehicleHalImpl: Received VehicleProperty: 0x21401245
LucidVehicleHalImpl: sending: InfoT_Body/CCC_HVAC_1/ICCC_HvacAirFlowSetFromHmiRearLe 65535.000000
LucidVehicleHalImpl: Received VehicleProperty: 0x21401247
LucidVehicleHalImpl: sending: InfoT_Body/CCC_HVAC_1/ICCC_HvacAirFlowSetFromHmiRearRi 65535.000000
LucidVehicleHalImpl: Received VehicleProperty: 0x21401243
LucidVehicleHalImpl: sending: InfoT_Body/CCC_HVAC_1/ICCC_HvacAirFlowSetFromHmiFrntRi 65535.000000

```

```
HvacController: Status Prop_ID 557847110 // 感觉这好像是另一个事件: CID 事件, 向下传 ??????????????????
HvacController: Status Prop_ID Value 70
LucidVehicleHalImpl: Received VehicleProperty: 0x21401241
HvacController: [HvacController.onChangeEvent():1258] handlePairedSignalsW() : bcmPropId=557847110, bcmValue=70, cachedValu
LucidVehicleHalImpl: sending: InfoT_Body/CCC_HVAC_1/ICCC_HvacAirFlowSetFromHmiFrntLe 65535.000000
HvacController: [HvacController.access$1800():29] mHvacListener.onUpdateFanSpeedW() : zone=34, fanSpeed=70
```

3 FrontSeatHvacController: Daemon / Client Interface 不太容易理解，多看几遍/想一想

- 一份他们起草过的架构文档：<https://lucidmotors.atlassian.net/wiki/spaces/INF0/pages/2609547231/CarServiceClient+WIP+Architecture+Design>

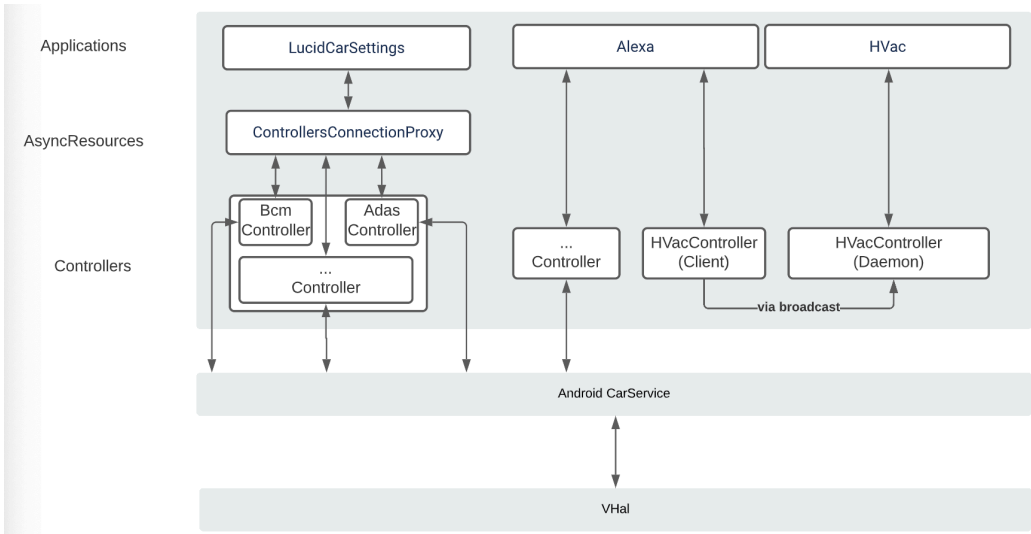


Figure 1: Existing Controller Architecture:

- For some controllers, they are using Demon / Client mechanism to communicate to keep one single source of truth. The performance would significantly drop down if it has too much messages between Demon and Client since they are using Intent as its communication channel.
- Controllers are not decoupling with UI(Activities / Fragment), it resulted in an issue that compilation would get failed once you change an interface of Controller. We can achieve this by using another decoupling solution.
- Android 10: Signal Controllers Refactoring (HvacController 和 FrontSeatHvacContoller 两者应该是属于最近刚 refactor 过的, <https://lucidmotors.atlassian.net/wiki/spaces/INF0/pages/2718205940/Android+10+Signal+Controllers+Refactoring>), 下面是一段比较帮助理解的内容：
 - For all incoming signals, applications will continue to directly communicate with VHAL to receive inbound signal updates. No additional code logic changes are necessary for this in Android 10. 自底层向 UI 传送的 status signal 是同以前一样，不曾变化的？这里自己也还是没有搞明白???????.....

- (Paul's approach) Split all controllers into client and daemon instances (比如这里他们刚 refactor 过的 HvacApp 中的 FrontSeatHvacController).

- * All signal controller implementations will be refactored and modified so they can be instantiated as either a daemon instance or as a client instance.
- * For every signal controller api that sends out a vehicle signal (request signal), the client instance will ask the daemon instance to perform the signal-send action on its behalf by sending a broadcast intent. The daemon instance will implement the BroadcastReceiver to receive the request, and handle implementation by sending signal to VHAL.

- 并且他们确实重构过了

Refactoring Existing Controllers

1. HvacController - (send & receive), **Daemon-Client Mode** (partially done by @Ray Wang)
2. AmbientLightController - (send & receive), **Daemon-Client Mode** (partially done by @Ray Wang)
3. FrontSeatHvacController - (send & receive), **Daemon-Client Mode** (partially done by @Ray Wang)
*** (bc used by Alexa / HVAC App)
4. <<----- for this section below, we need Daemon-Client Mode as well ----->>
5. AdasController - (send & receive)

```
//===== Daemon / Client Interface =====
private BroadcastReceiver mDaemonClientReceiver = new BroadcastReceiver() { // 这里就是 Daemon Controller 接收 Client Control
    @Override public void onReceive(Context context, Intent intent) { // 广播来自 Client, 由 Daemon 代为与 VHAL 交互处理
        String action = intent.getAction();
        if (ACTION_ADJUST_CUSHION_COOLING.equals(action)) {
            SeatHVACLevel seatHvacLevel = (SeatHVACLevel) intent.getSerializableExtra(EXTRA_ADJUST_CUSHION_COOLING);
            int whichSeat = intent.getIntExtra(EXTRA_ADJUST_CUSHION_COOL_SEAT, 0);
            Logger.d(TAG, "OnReceive, adjustCushionCooling() whichSeat : " + whichSeat + " Seat Hvac Level " + seatHvacLevel);
            adjustCushionCooling(whichSeat, seatHvacLevel);
        } else if (ACTION_ADJUST_CUSHION_HEATING.equals(action)) {
            SeatHVACLevel seatHvacLevel = (SeatHVACLevel) intent.getSerializableExtra(EXTRA_ADJUST_CUSHION_HEATING);
            int whichSeat = intent.getIntExtra(EXTRA_ADJUST_CUSHION_HEATING_SEAT, 0);
            Logger.d(TAG, "OnReceive, adjustCushionHeating() whichSeat : " + whichSeat + " Seat Hvac Level " + seatHvacLevel);
            adjustCushionHeating(whichSeat, seatHvacLevel);
        } else if (ACTION_ADJUST_BACKREST_HEATING.equals(action)) {
            SeatHVACLevel seatHvacLevel = (SeatHVACLevel) intent.getSerializableExtra(EXTRA_ADJUST_BACKREST_HEATING);
            int whichSeat = intent.getIntExtra(EXTRA_ADJUST_BACKREST_HEATING_SEAT, 0);
            Logger.d(TAG, "OnReceive, adjustBackrestHeating() whichSeat : " + whichSeat + " Seat Hvac Level " + seatHvacLevel);
            adjustBackrestHeating(whichSeat, seatHvacLevel);
        } else if (ACTION_ADJUST_BACKREST_COOLING.equals(action)) {
            SeatHVACLevel seatHvacLevel = (SeatHVACLevel) intent.getSerializableExtra(EXTRA_ADJUST_BACKREST_COOLING);
            int whichSeat = intent.getIntExtra(EXTRA_ADJUST_BACKREST_COOLING_SEAT, 0);
            Logger.d(TAG, "OnReceive, adjustBackrestCooling() whichSeat : " + whichSeat + " Seat Hvac Level " + seatHvacLevel);
            adjustBackrestCooling(whichSeat, seatHvacLevel);
        }
    }
};

private static final String ACTION_ADJUST_CUSHION_COOLING = "com.lucid.car.carctrl.ACTION_ADJUST_CUSHION_COOLING";
private static final String EXTRA_ADJUST_CUSHION_COOLING = "com.lucid.car.carctrl.EXTRA_ADJUST_CUSHION_COOLING";
private static final String EXTRA_ADJUST_CUSHION_COOL_SEAT = "com.lucid.car.carctrl.EXTRA_ADJUST_CUSHION_COOL_SEAT";
private void adjustCushionCoolingClient(int whichSeat, SeatHVACLevel requestCushionCool) {
    Logger.d(TAG, "adjustCushionCoolingClient()");
    if (mContext == null) {
        Logger.e(TAG, "Context is null, broadcast cannot be sent");
        return;
    }
    Intent intent = new Intent(ACTION_ADJUST_CUSHION_COOLING);
    intent.putExtra(EXTRA_ADJUST_CUSHION_COOLING, requestCushionCool);
    intent.putExtra(EXTRA_ADJUST_CUSHION_COOL_SEAT, whichSeat);
    // 应用层所有应用中, 某个唯一存在的同类 DaemonController 会接收到应用层的这些广播, 并为其代向硬件申请调值
    mContext.sendBroadcast(intent);
}
/**
```

```

* Cushion Cooling
* Driver Seat = 1; Passenger Seat = 2;
* off=0, low=1, medium= 2, high= 3
*/
public void adjustCushionCooling(int whichSeat, SeatHVACLevel requestCushionCool) { // 由 UI 向下 / 向硬件传值的执行函数
    if (!mIsSignalDaemon) { // client:
        adjustCushionCoolingClient(whichSeat, requestCushionCool); // <<<<===== 方法定义在上面 ^|
        return;
    }
    // Edge case to check if requested cool level is null.
    if (requestCushionCool == null) {
        Logger.d(TAG, "Invalid Cushion Cool Level");
        return;
    }
    if (whichSeat == 0) {
        Logger.d(TAG, "Not a valid seat");
        return;
    }
    int value = 0;
    // Converting requested cushion cool level to int, which can be sent to BCM.
    switch (requestCushionCool) {
        case OFF:
            value = 0;
            break;
        case LOW:
            value = 1;
            break;
        case MEDIUM:
            value = 2;
            break;
        case HIGH:
            value = 3;
            break;
    }
    Logger.d(TAG, "Controller : Requested Cushion Cool value " + value + " for Seat " + whichSeat);
    if (whichSeat == DRIVER_SEAT) {
        // Caching driver cushion cool.
        synchronized (mPropertiesLock) { // 同步重置：更新当前 DaemonController 中所存储的这一属性的状态值
            mDriverCushionCool = requestCushionCool;
        }
        // 如果不久前有发送过同一属性的消息，会将它们全部取消：都当作过时处置，如果多个应用多层输入有竞争，最新更新永远胜。。。

        mSignalBroadcastHandler.removeMessages(ISEAT_LF_CUSHION_FAN_CMD);
        // 向下传递：通过消息机制向下向底层 BCM 发送（要求调整硬件值的）消息
        sendSignalToBCM(ISEAT_LF_CUSHION_FAN_CMD, value);
        // 下面，延迟 1 秒：是否，可以假定刚才，上面一行，向硬件发送的消息，正常情况下，应该能够在 1 秒内处理完成？
        Message message = mSignalBroadcastHandler.obtainMessage(ISEAT_LF_CUSHION_FAN_CMD, mDriverCushionCoolRunnable);
        mSignalBroadcastHandler.sendMessageDelayed(message, TIME_OUT_IN_MILLIS);
    }
    // 只要 UI 回调非空，同时向 UI 层同步当前值（过程可能多余，可是异常总是无处不在 ???）
    if (mFrontSeatHvacListener != null) {
        mFrontSeatHvacListener.onUpdateCushionCoolingW(DRIVER_SEAT, requestCushionCool);
    }
} else {
    // Caching passenger cushion cool. ...
}
}

private static final int TIME_OUT_IN_MILLIS = 1000;
// Helper method to send signal to BCM:
private void sendSignalToBCM(int bcmPropId, int bcmValue) { // 向下传递：通过消息机制向下向底层 BCM 发送（要求调整硬件值的）消息
    mSignalBroadcastHandler.sendMessageDelayed( // 延迟发送消息
        mSignalBroadcastHandler.obtainMessage(MSG_BROADCAST_SIGNAL, bcmPropId, bcmValue), 10);
}

// Driver cushion cool runnable: 由应用层向下向硬件申请调值是一回事，还有其它可能：
// 是否有车上同类物理键输入/发送的消息处理过程中是否有意外？（其它可能，还有吗 ???）
final TimeoutRunnable mDriverCushionCoolRunnable = new TimeoutRunnable() {
    @Override
    public void run() {
        Logger.d(TAG, "Cushion Time Out Runnable");
        int driverCushionCoolStatusValue = 0;
        try {
            if (mPropertyManager != null) // 从硬件 FW 中索取 Status Signal 当前最新状态值
                driverCushionCoolStatusValue = mPropertyManager.getIntProperty(ISEAT_LF_CUSHION_FAN_STS, 0);
        } catch (CarNotConnectedException e) {
            e.printStackTrace();
        }
    }
}

```

```

        boolean updateCallBack = false;
        SeatHVACLevel cushionCoolLevel = intToHvacLevelEnum(driverCushionCoolStatusValue);
        synchronized (mPropertiesLock) {
// 如果 当前 DaemonController 中最近更新值 与 硬件最新状态值 不同
            if (mDriverCushionCool != cushionCoolLevel) { // 当前 DaemonController 中 request 值与硬件 status 值不同
                // Matching request signal with status signal,
// 如果当前 DaemonController 中 request 值与硬件 status 值不同 (原因? 异常), 永远以硬件 status 值为准, UI 不能错误显示硬件当前状态
                mDriverCushionCool = cushionCoolLevel; // 将当前 request 值同步为硬件最新状态 status 值
// XXXX: (消息机制 mQueue 里的消息还没有处理?) 向下向硬件再次发送消息, 以清除 mQueue 里可能存在的未处理消息, 使刚索取的 status 值能
// 这里自己想得不对: 在 vhal 层面, 仍然也有一个状态值, 虽然 requestSignal 发送过程中出了意外, 没能修改成功
// 但是 vhal 层面的这个属性值被同步成了我们的 requestSignal 想要的值, 还需要再发送一条消息把这个属性状态值同步为硬件属性值
                sendSignalToBCM(ISEAT_LF_CUSHION_FAN_CMD, driverCushionCoolStatusValue);
                updateCallBack = true; // 同时激活向 UI 的回调: 要求 UI 与 status signal 值同步
            }
        }
        if (updateCallBack && (mFrontSeatHvacListener != null)) // 只在必要时, 重置 UI:
            // 这是因为当 UI 发出 request 的时候, UI 自己自动更新为自己想要成为的状态了; 但是现在更新失败了, 那么 UI 就务必再重新请求
            mFrontSeatHvacListener.onUpdateCushionCoolingW(DRIVER_SEAT, cushionCoolLevel); // 从底向上传, 回调
    }
};
abstract static class TimeoutRunnable implements Runnable {} // 应用是在实现类中, 有相应的 timeout 机制, Ui 发出 request 之后,
//=====
// Handling Broadcasting of Signals (from CCC)
//=====
private static final int MSG_BROADCAST_SIGNAL = 100;
int[] input = new int[3];
private HandlerThread mHandlerThread;
private SignalBroadcastHandler mSignalBroadcastHandler;

class SignalBroadcastHandler extends Handler {
    SignalBroadcastHandler(Looper looper) {
        super(looper);
    }
    @Override
    public void handleMessage(Message msg) {
        if (msg.what == MSG_BROADCAST_SIGNAL) {
            broadcastCccSignalW(msg.arg1, msg.arg2);
        } else if (msg.obj instanceof TimeoutRunnable) { // Executing (timeout) runnable.
            TimeoutRunnable timeOutRunnable = (TimeoutRunnable) msg.obj;
            timeOutRunnable.run();
        }
    }
    void broadcastCccSignalW(int propId, int value) { // 向下传递: 由 UI 向硬件设置属性值
        try {
// 搞不懂这个 mPropertyManager 是从哪里来的: 总之就是一个车硬件属性状态的总管, 一车一个
            if (mPropertyManager != null) {
                mPropertyManager.setIntProperty(propId, 0, value);
                mPropertyManager.setProperty(int[].class, 1, 0, new int[]{1, 2, 3});
            }
        } catch (android.car.CarNotConnectedException e) {
            Logger.d(TAG, "broadcastCccSignalW(): VHAL not connected.");
        }
    }
}
// Incoming Signals Handling: 由硬件向 UI 传值的回调接口
private CarPropertyManager.CarPropertyEventCallback mPropertiesListener = new CarPropertyManager.CarPropertyEventCallback()
    @Override
    public void onErrorEvent(int i, int i1) {
        Logger.d(TAG, "onErrorEvent(): i=" + i + ", i1=" + i1);
    }
    @Override // 当底层硬件的值发生变化的时候, 向上向 UI 方向传递的回调函数
    public void onChangeEvent(CarPropertyValue carPropertyValue) {}
};

```

3.1 关于车的来源: 每辆 Lucid 车整个系统只有一个 Car: Car vs mProperty-Manager

- 他们 refactor, 把这辆车/把 Controllers 可能用到的公共部分, 都提取放在 ControllerBase 里面了

3.1.1 ControllerBase.java

```
public abstract class ControllerBase<T> extends ControllerListenerBase<T> implements OnCarServiceStatusChangeListener {

    private LucidCar mLucidCar;
    private List<Integer> mStatusSignals;
    private Set<Integer> mStatusSignalValueIndicator;
    private boolean mSubscribeSuccessfully;
    private Object mLock;
    protected LucidCarPropertyManager mPropertyManager;
    protected T mControllerListener;
    protected String TAG;

    private CarPropertyManager.CarPropertyEventCallback mCarPropertyEventCallback = new CarPropertyManager.CarPropertyEventCallback() {
        @Override public void onChangeEvent(CarPropertyValue carPropertyValue) {
            int propertyId = carPropertyValue.getPropertyId();
            checkIfNotifyStatus(propertyId); // 检查一下是否需要回传状态: 可能有某些情况下是不需要再返回状态的
            ControllerBase.this.onChangeEvent(carPropertyValue);
        }
    };
    @Override public void onErrorEvent(int i, int il) { }
};

@CallSuper public void init(@NonNull Context context, T controllerListener) { // 两种 init(listener) 方法都是需要传回调的
    mControllerListener = controllerListener;
    TAG = this.getClass().getSimpleName();
    mLock = new Object();
    mLucidCar = LucidCarProvider.getLucidCar(context);
    mPropertyManager = mLucidCar.mPropertyManager;
    mStatusSignals = getStatusSignals();
    if (mStatusSignals == null)
        mStatusSignals = new ArrayList<>(); // 这里是一个链表 ArrayList: 对于每个控制器来说, 它所监控的可能都是一组 carProperty
    mStatusSignalValueIndicator = new HashSet<>();
    if (mLucidCar.mCarServiceStatus == CarServiceStatus.CONNECTED) {
        mSubscribeSuccessfully = true;
        subscribe();
        if (mControllerListener != null)
            mControllerListener.onVhalConnectedW();
    }
    mLucidCar.setCarServiceStatusChangeListener(this);
}

@CallSuper public void init(T controllerListener) { // 两种 init(listener) 方法都是需要传回调的
    mControllerListener = controllerListener;
    unsubscribe();
    subscribe();
}

@Override @CallSuper // 在进行所有的连接工作之前, 首先需要确定, 跟车是否已经连起来了?
    public void onCarServiceStatusChange(CarServiceStatus status) {
        if (status == CarServiceStatus.CONNECTED && !mSubscribeSuccessfully) {
            subscribe();
            if (mControllerListener != null) // backward compatible code, for Android 9 Controllers? 这些理解对吗?
                mControllerListener.onVhalConnectedW();
        }
        if (status == CarServiceStatus.DISCONNECTED) {
            mStatusSignalValueIndicator.clear();
            unsubscribe();
            mSubscribeSuccessfully = false;
            if (mControllerListener != null) // backward compatible code
                mControllerListener.onVhalDisconnectedW();
        }
        mPropertyManager = mLucidCar.mPropertyManager; // 给这个变量赋值, 以便不曾重构的控制器仍然能够正常工作, mPropertyManager
    }

@Override public void onLucidCarPropertyManagerReady(LucidCarPropertyManager lucidCarPropertyManager) {
    mPropertyManager = lucidCarPropertyManager; // 如果之前是 null 的话, 现在可以得到非空的值
}

abstract void onChangeEvent(CarPropertyValue carPropertyValue); // carPropertyValue = carPropertyId
abstract List<Integer> getStatusSignals(); // 这里是一个链表 ArrayList<>()

@CallSuper protected void cleanup() {
    mLucidCar.removeCarServiceStatusChangeListener(this);
    if (mStatusSignals != null) {
        mStatusSignals.clear();
        mStatusSignals = null;
    }
    if (mStatusSignalValueIndicator != null) {
        mStatusSignalValueIndicator.clear();
        mStatusSignalValueIndicator = null;
    }
}
```

```

    }
}
private void subscribe() {
    mStatusSignalValueIndicator.clear();
    mStatusSignalValueIndicator.addAll(mStatusSignals); // 赋值
    mLucidCar.subscribe(mStatusSignals, mCarPropertyEventCallback);
}
private void unsubscribe() {
    if (mStatusSignals != null)
        mLucidCar.unsubscribe(mStatusSignals, mCarPropertyEventCallback);
}
private void checkIfNotifyStatus(int propertyId) {
    if (mStatusSignalValueIndicator != null && mStatusSignalValueIndicator.size() == 0) {
        if (mControllerListener != null) {
            // backward compatible code
            mControllerListener.onPropertiesReadyW();
            mControllerListener.onCarServiceStatusChangedW(CarServiceStatus.PROPERTY_VALUE_READY);
        }
    } else {
        synchronized (mLock){
            mStatusSignalValueIndicator.remove(propertyId);
        }
    }
}
}
}
}
}

```

3.1.2 LucidCarService app: 在这里其实是没有任何必要的，暂时先放这里

- http://hqdevbld01.corp.lucid.lcl:8888/source/xref/lucid_dash5dv_10r40/vendor/lucid/proprietary/apps/LucidCarService/app/src/main/java/com/lucid/car/service/powerstates/PowerStateController.java

```
package com.lucid.car.service.powerstates;
```

```

public class PowerStateController {
    private static final String TAG = "PowerStateController";

    public static final int POWER_STATE_UNDEF = -1;
    public static final int POWER_STATE_SLEEP = 0;
    public static final int POWER_STATE_WINK = 1;
    public static final int POWER_STATE_ACCESSORY = 2;
    public static final int POWER_STATE_DRIVE = 3;
    public static final int POWER_STATE_LIVE_CHARGE = 4;
    public static final int POWER_STATE_SLEEP_CHARGE = 5;
    public static final int POWER_STATE_LIVE_UPDATE = 6;
    public static final int POWER_STATE_SLEEP_UPDATE = 7;
    public static final int POWER_STATE_CLOUD1 = 8;
    public static final int POWER_STATE_CLOUD2 = 9;
    public static final int POWER_STATE_MONITOR = 10;

    public static class PowerStateListener {
        void onVHalConnectedW() {};
        void onVHalDisconnectedW() {};
        void onUpdatePowerStateW(int powerState) {};
    }

    //=====
    // Init / Cleanup
    //=====
    Context mContext;
    PowerStateListener mPowerStateListener;

    public void init(Context context, PowerStateListener listener) {
        mContext = context;
        mPowerStateListener = listener;
        mCarApiClient = Car.createCar( mContext, mCarConnectionCallback );
        mCarApiClient.connect();
    }
    public void cleanup() {
        if ( mCarApiClient != null ) {
            mCarApiClient.disconnect();
        }
    }
}

```

```
mPowerStateListener = null;
mContext = null;
}

//=====
// Connection to VHal
//=====
private Car mCarApiClient;
// 可惜：我要寻找的是 mPropertyManager !!! 可是，它们虽然名字不同，可能指的却是同一个东西 ???!!
// （因为每个 Lucid 都只有一辆车 Car，而这个车的 PropertyManager 也应该 有且仅有一个 ? ），可是我的假设需要代码源码理论支持.....
private CarPropertyManager mPropertiesManager;
}
```

```

        mMessages = msg.next;
        msg.next = null;
        if (DEBUG) Log.v(TAG, "Returning message: " + msg);
        msg.markInUse();
        return msg;
    }
} else {
    // No more messages.
    nextPollTimeoutMillis = -1;
}
}
}

```

- 很贴心的给出了注释解释 “Next message is not ready. Set a timeout to wake up when it is ready.”，翻译 “下一条消息尚未准备好。设置一个超时，以便在准备就绪时唤醒。”
- when 就是 uptimeMillis，for (;;) 相当于 while(true)，如果头部的这个 Message 是有延迟而且延迟时间没到的 (now < msg.when)，不返回 message（不会把它重新放回 mQueue 中，要不然你把它放回后，你如何保证在闹钟叫醒的时候一定是它是 mQueue 的队头可以取得出来，再要不然消息不在队头的时候要怎么才能从 mQueue 中把想要的消息取出来呢？）而且会计算一下时间（保存为变量 nextPollTimeoutMillis），然后再循环的时候判断如果这个 Message 有延迟，就调用 nativePollOnce(ptr, nextPollTimeoutMillis) 进行阻塞。nativePollOnce() 的作用类似与 object.wait()。得出结论是通过阻塞实现的。

4.2.1 如果在延迟消息还没有唤醒时又有新的消息进来，会怎样处理呢，看 enqueueMessage() 方法源码

```

boolean enqueueMessage(Message msg, long when) {
    if (msg.target == null)
        throw new IllegalArgumentException("Message must have a target.");
    if (msg.isInUse())
        throw new IllegalStateException(msg + " This message is already in use.");
    synchronized (this) {
        if (mQuitting) {
            IllegalStateException e = new IllegalStateException(
                msg.target + " sending message to a Handler on a dead thread");
            Log.w(TAG, e.getMessage(), e);
            msg.recycle();
            return false;
        }
        msg.markInUse();
        msg.when = when;
        Message p = mMessages; // p: curr msg: 可能为空，如果当前 mQueue 里还没有任何消息的话
        boolean needWake;
        if (p == null || when == 0 || when < p.when) { // 要将想入队列的消息放队头
            // New head, wake up the event queue if blocked.
            msg.next = p;
            mMessages = msg;
            needWake = mBlocked; // mBlocked: 在 next() 中当消息为延迟消息的时候 mBlocked=true
        } else { // 不能放在队头：需要帮它定位到它应该在的位置
            // Inserted within the middle of the queue. Usually we don't have to wake
            // up the event queue unless there is a barrier at the head of the queue
            // and the message is the earliest asynchronous message in the queue.
            needWake = mBlocked && p.target == null && msg.isAsynchronous();
            Message prev;
            for (;;) {
                prev = p;
                p = p.next;
                if (p == null || when < p.when) // mQueue 是一个 PriorityQueue，按时间顺序排列由小到大排列的吗？
                    break; // 如果是这样，那么找到合适的旋转位置，就退出循环
                if (needWake && p.isAsynchronous())
                    needWake = false;
            }
            msg.next = p; // 把当前消息 msg 放在 prev 和 p 的中间：invariant: p == prev.next
            prev.next = msg;
        }
        // We can assume mPtr != 0 because mQuitting is false.
        if (needWake)
            nativeWake(mPtr);
    }
}

```

```

    return true;
}

```

- 在这里 `p` 是现在消息队列中的头部消息，我们看到 | `when < p.when` 的时候它交换了放入 `message` 与原来消息队列头部 `P` 的位置，并且 `needWake = mBlocked`; (在 `next()` 中当消息为延迟消息的时候 `mBlocked=true`), 继续向下看当 `needWake =true` 的时候 `nativeWake(mPtr)` (唤起线程)
- 一切都解释的通了，如果当前插入的消息不是延迟 `message`，或比当前的延迟短，这个消息就会插入头部并且唤起线程来

4.3 安卓 `Handler.removeMessages(int what)`

```

public final void removeMessages(int what) {
    mQueue.removeMessages(this, what, null);
}

```

- 用于移除一组 `Message`，`Message` 持有的 `int` 值 `what` 只要匹配，都会被移除掉。传入参数为 `int` 值，表示 `Message` 对象持有的一个 `int` 值
 - 1、传入的局部变量 `what` 持有的 `int` 值最终会传入到 `Handler` 对象持有的 `MessageQueue` 对象 `mQueue` 的 `removeMessages()` 方法中
 - 2、`removeMessages` 接受三个参数，第一个参数是当前的 `Handler` 对象，第二个参数是当前传入的整型 `what` 值，最后一个参数这里传入的是 `null`，该参数接受的类型是 `Object`。
 - 3、`MessageQueue` 会将消息队列中，所有 `what` 值匹配的未分发出去的 `Message` 对象全部移除掉

4.4 `Handler.removeMessages` 的作用，有时候为什么一定要先 `remove` 一下呢

- `removeMessages` 会将 `handler` 对应 `message queue` 里的消息清空，如果带了 `int` 参数则是对应的消息清空。队列里面没有消息则 `handler` 会不工作，但不表示 `handler` 会停止。当队列中有新的消息进来以后 `handler` 还是会处理。
- 我的理解：

- 1、这个方法使用的前提是之前调用过 `sendEmptyMessageDelayed(0, time)`，意思是延迟 `time` 执行 `handler` 中 `msg.what=0` 的方法；

```

// Set 750ms timeout timer.
if ( mHvacSignalsTimeoutMillis > 0 ) { // 延迟 mHvacSignalsTimeoutMillis 这么长时间，再去执行这个属性的所有 ??? 消息
    mSignalBroadcastHandler.sendEmptyMessageDelayed( bcmPropId, mHvacSignalsTimeoutMillis );
}

```

- 2、在延迟时间未到的前提下，执行 `removeMessages(0)`，则上面的 `handler` 中 `msg.what=0` 的方法取消执行；
- 3、在延迟时间已到，`handler` 中 `msg.what=0` 的方法已执行，再执行 `removeMessages(0)`，不起作用。

5 `HvacPresenter.java`: 这里之前自己写的是，现在的自己根本都快看不懂了呀

```

public class HvacPresenter extends HvacController.HvacListener implements PresenterContract {
    // HvacController.HvacListener: 向上传递，根据硬件的状态，向上向 UI 传递信号的回调
}

```

```

public static class HvacListener extends ControllerListenerBase {
    public void onUpdatePowerW(boolean isPowerOn, boolean isPrecondOn, boolean isKeepTempOn) { }
    public void onUpdateAcW(boolean isAcOn) { }
    public void onUpdateAirCircW(int airCircMode) { }
    public void onUpdateDefrostW(boolean isFrontDefrostOn) { }
    public void onUpdateSyncZonesW(boolean isSyncOn) { }
    public void onUpdateAutoW(int zone, boolean isAutoOn) { }
    public void onUpdateVentModeW(int zone, boolean isVentUpOpen, boolean isVentMidOpen, boolean isVentDownOpen) { }
    public void onUpdateFanSpeedW(int zone, int speed) { }
    public void onUpdateTemperatureW(int zone, int temp) { }
    public void onUpdateRearPowerW(int whichSeat, boolean isRearPowerOn) { }
    public void onUpdateRearDefrostW(boolean isRearDefrostOn) { }
    public void onUpdateSteeringHeat(boolean isSteeringHeatOn) { }
}

// PresenterContract: 向下传递, 根据 UI 点击/滑动等事件, 向下向底层回调, 索求底层 FW/HW 硬件与 UI 同步
public interface PresenterContract {
    void hvacPowerSwitchEvent(Boolean isON);
    void hvacRearPowerSwitchEvent(Boolean isON);
    void hvacACSwitchEvent(Boolean isON);
    void hvacAirFlowSwitchEvent();
    void hvacFanSliderEvent(SliderData mFanSliderData);
    void hvacTempSliderEvent(SliderData mTempSliderData);
    void hvacAirFlowDirectionEvent(VentModeData ventModeData);
    void hvacSyncEvent(Boolean isEnabled);
    void hvacAutoEvent(AutoState autoState);
    void hvacFrontDefrostEvent(Boolean state);
    void hvacRearDefrostEvent(Boolean state);
    boolean checkIsDefrostON();
    void onResumeEvent();
    void hvacAutoConditionEvent(Boolean state);
    Boolean checkAutoConditionState();
    void hvacKeepTemperatureEvent(Boolean state);
    Boolean checkKeepTempState();
    Boolean isBatteryLevelDown();
    void hvacSteeringHeatEvent(Boolean state);
    boolean checkIsSyncOn();
    boolean checkHvacOn();
    boolean checkRearOn();
    int getFanZone(int zone);
    boolean checkZoneOffWindshieldOn(int requestZone);
}

// ViewContract: 向上传递, 从 HvacPresenter 根据 FW/HW 状态向上更新 UI 的调用方法
public interface ViewContract {
    void updateHVACPowerUIState(Boolean isOn);
    void updateHvacACUIState(Boolean isOn);
    void updateHvacRearPowerUIState(Boolean isOn);
    void updateHvacAirCircUIState(int airCircState);
    void updateHvacSyncUIState(Boolean isSyncModeOn);
    void updateHvacAutoUIState(int zone, Boolean state);
    void updateHvacVentUIState(int zone, VentStates ventStates);
    void passFanZoneChange(int zone, int speed, boolean isHVacPowerOn);
    void passTemZoneChange(int zone, float temp);
    void updateFanVisibilityStatus(int zone);
    void updateDefrostChange(boolean state);
    void updateRearDefrostChange(boolean state);
    void undoSyncState();
    void onUpdateAutoConditionState(boolean state);
    void onUpdateKeepTemperature(boolean state);
    void onUpdateSteeringHeatState(boolean state);
}

```

- 构造器

```

public HvacPresenter(AppCompatActivity activity, HvacController hvacController) {
    this.mContext = activity;
    this.viewContract = (ViewContract) activity; //
    mHvacController = hvacController;
    registerHandlerEvents();
}

private void registerHandlerEvents() {
    uiHandler = new Handler(Looper.getMainLooper(), msg -> {
        //Logger.d(TAG, "in ui handler" + msg.what);
        switch (msg.what) {
            case PROPERTY_READY: {
                handleInitialPropertySignal();
            }
        }
    });
}

```

```

        }
        break;
    }
}

private void handleInitialPropertySignal() {
    isHvacInitized = true;
    manuallyGetUpdatedValues();
    autoConditionInit();
}

private void manuallyGetUpdatedValues() {
    if (mHvacController != null) {
        boolean isPowerOn = mHvacController.isPowerOn();
        boolean isPrecondOn = mHvacController.isPreconditioningOn();
        boolean isKeepTempOn = mHvacController.isKeepTempOn();
        Logger.d(TAG, "Is Precondition ON: " + isPrecondOn + "Power ON state: " + isPowerOn + "IS Keep temperature state :");
        handlePowerAndPreConditionSignal(new PowerStates(isPowerOn, isPrecondOn, isKeepTempOn));
    }
}

private void handlePowerAndPreConditionSignal(final Object obj) {
    try {
        if (obj instanceof PowerStates) {
            final PowerStates powerStates = ((PowerStates) obj); // 从 FW/HW 获取当前状态, 然后向上更新 UI
            Boolean power = powerStates.getPowerOn();
            Boolean preConditionState = powerStates.getPreconditionOn();
            Boolean keepTempState = powerStates.getKeepTempOn();
            viewContract.updateHVACPowerUIState(power); // viewContract
            viewContract.onUpdateKeepTemperature(keepTempState);
            Logger.d(TAG, "Check 2 in POWER_STATE = " + power + " to see HVAC and PRECONDITION SIGNALS verification");
            // Logger.d(TAG, "Check 2 in PreConditionSignal = " + preConditionState + " to see HVAC and PRECONDITION SIG");
            // Logger.d(TAG, "Check 3 in KeepTempSignal = " + keepTempState + " to see HVAC and KEEP TEMP SIGNALS verifi");
        }
    } catch (Exception e) {
        //
    }
}
}

```

6 PresenterContract: 根据 UI 点击/滑动等事件, 向下向底层回调, 索求底层 FW/HW 硬件与 UI 同步

```

void hvacPowerSwitchEvent(Boolean isON);
void hvacRearPowerSwitchEvent(Boolean isON);
void hvacACSwitchEvent(Boolean isON);
void hvacAirFlowSwitchEvent();
void hvacFanSliderEvent(SliderData mFanSliderData);
void hvacTempSliderEvent(SliderData mTempSliderData);
void hvacAirFlowDirectionEvent(VentModeData ventModeData);
void hvacSyncEvent(Boolean isEnabled);
void hvacAutoEvent(AutoState autoState);
void hvacFrontDefrostEvent(Boolean state);
void hvacRearDefrostEvent(Boolean state);
boolean checkIsDefrostON();
void onResumeEvent();
void hvacAutoConditionEvent(Boolean state);
boolean checkAutoConditionState();
void hvacKeepTemperatureEvent(Boolean state);
boolean checkKeepTempState();
boolean isBatteryLevelDown();
void hvacSteeringHeatEvent(Boolean state);
boolean checkIsSyncOn();
boolean checkHvacOn();
boolean checkRearOn();
int getFanZone(int zone);
boolean checkZoneOffWindshieldOn(int requestZone);

```

- 因为看不懂, 拿一两个方法来分析一下:

```

void hvacFanSliderEvent(SliderData mFanSliderData);
boolean checkHvacOn();

```

- 实现代码

```

public void hvacFanSliderEvent(SliderData mFanSliderData) {
    if (mHvacController != null) { // 仍然是先从三件获取当前 (FW/HW 硬件) 最新状态
        Logger.d(TAG, "FAN SLIDER is SYNC mode: " + mHvacController.isSyncZones() + " is AUTO ON: " +
            mHvacController.isAutoOn(mFanSliderData.getZone()) + " ZONE is: " + mFanSliderData.getZone());
        if (!checkHvacInit()) {
            return;
        }
    }
    checkAndSetHvacOn();
    checkAndSetDefrost();
    checkAndSetRearHvacOn(mFanSliderData.getZone(), mFanSliderData.getmValue());
    if (mHvacController != null) {
        if (mHvacController.isSyncZones()) {
            if (mFanSliderData.getZone() == ZONE_FRONT_LEFT) {
                sendFanValueToController(ZONE_FRONT_LEFT, mFanSliderData.getmValue()); //
            } else {
                mHvacController.setSyncZones(false);
                viewContract.undoSyncState();
                sendFanValueToController(mFanSliderData.getZone(), mFanSliderData.getmValue());
            }
        }
        if (mHvacController.isAutoOn(mFanSliderData.getZone())) {
            mHvacController.setAutoOn(mFanSliderData.getZone(), false);
            viewContract.updateFanVisibilityStatus(mFanSliderData.getZone());
            sendFanValueToController(mFanSliderData.getZone(), mFanSliderData.getmValue());
        } else {
            viewContract.updateFanVisibilityStatus(mFanSliderData.getZone());
            sendFanValueToController(mFanSliderData.getZone(), mFanSliderData.getmValue());
        }
    }
}

private void sendFanValueToController(int zone, float appFanValue) { // 根据 UI Slider 滑动得到的值, 去通过 HvacController 来
    int mappedFanSpeed = mapFanSpeed(0, 10, 0, 100, Math.round(appFanValue));
    //Logger.d(TAG, "ZONE: " + zone + " Original Speed: " + appFanValue + " Mapped speed: " + mappedFanSpeed);
    if (mHvacController != null) {
        mHvacController.setFanSpeed(zone, mappedFanSpeed);
    }
}

public boolean checkHvacOn() { // 查询中间层/底层 FW/HW 的最新物理状态, 来向上, 向下???
    if (mHvacController != null) {
        return (checkHvacInit() && mHvacController.isPowerOn());
    } else {
        return false;
    }
}
}

```

7 FrontViewFragment:

```

public class FrontViewFragment extends Fragment
    implements SeatClimateLevel.SeatClimateLevelChangeListener, View.OnClickListener {
    private FrontSeatHvacController mFrontSeatHvacController;
    private FrontSeatHvacListener mFrontSeatHvacListener;

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
        Bundle savedInstanceState) {
        super.onCreateView(inflater, container, savedInstanceState);
        View view = inflater.inflate(R.layout.fragment_front_layout, container, false);
        mProfileData = new ProfileData(view.getContext());
        // ..... findViewById() s
        mDriverSeatHeater.setLevelChangeListener(this);
        mDriverSeatCooler.setLevelChangeListener(this);
        mPassengerSeatHeater.setLevelChangeListener(this);
        mPassengerSeatCooler.setLevelChangeListener(this);
        // ..... findViewById() s
        mSteeringHeatButton.setOnClickListener(this);
        setClickListener();
        initSliderCallbacks();

        mFrontSeatHvacListener = new FrontSeatHvacListener();
        mFrontSeatHvacController.init(getContext(), mFrontSeatHvacListener);
    }
}

```

```

// 设置当前 mFrontSeatHvacController 为所有应用层应用 (apps: Alexa, etc) 的 DaemonController
// 1. 将它自己设置为来自所有应用层 ClientController 应用 apps 的广播信号的 BroadcastReceiver 接收者
// 2. 需要为所有 FrontSeatHvacController 接收到的应用层 request 向硬件申请调值
mFrontSeatHvacController.setSignalDaemon(true);

Activity activity = getActivity();
if( activity instanceof HvacActivity ) {
    ((HvacActivity) activity).showSecondFragment();
}
return view;
}
}

```

8 FrontSeatHvacController

8.1 abstract ControllerBase<T extends ControllerListenerBase> class

```

public abstract class ControllerBase<T extends ControllerListenerBase> implements OnCarServiceStatusChangeListener {
    private LucidCar mLucidCar;
    private List<Integer> mStatusSignals;
    private Set<Integer> mStatusSignalValueIndicator;
    private boolean mSubscribeSuccessfully;
    private Object mLock;
    protected LucidCarPropertyManager mPropertyManager;
    protected T mControllerListener;
    protected String TAG;

    private CarPropertyManager.CarPropertyEventCallback mCarPropertyEventCallback
    = new CarPropertyManager.CarPropertyEventCallback() {
        @Override
        public void onChangeEvent(CarPropertyValue carPropertyValue) {
            int propertyId = carPropertyValue.getPropertyId();
            checkIfNotifyStatus(propertyId); // 有时候, 可能是不需要再继续向上向 UI 层传递信号的
            ControllerBase.this.onChangeEvent(carPropertyValue);
        }
        @Override
        public void onErrorEvent(int i, int i1) { }
    };
    @CallSuper
    public void init(@NonNull Context context, T controllerListener) {
        mControllerListener = controllerListener;
        TAG = this.getClass().getSimpleName();
        mLock = new Object(); // 同步锁
        mLucidCar = LucidCarProvider.getLucidCar(context); // 整个系统一辆车?
        mPropertyManager = mLucidCar.mPropertyManager; //
        mStatusSignals = getStatusSignals();
        if (mStatusSignals == null)
            mStatusSignals = new ArrayList<>();
        mStatusSignalValueIndicator = new HashSet<>();

        if (mLucidCar.mCarServiceStatus == CarServiceStatus.CONNECTED) { //
            mSubscribeSuccessfully = true;
            subscribe();
            if (mControllerListener != null)
                mControllerListener.onVhalConnectedW();
        }
        mLucidCar.setCarServiceStatusChangeListener(this);
    }
    @CallSuper
    public void init(T controllerListener){
        mControllerListener = controllerListener;
        unsubscribe(); // 怎么会同时调用这两个的?
        subscribe(); // 怎么会同时调用这两个的?
    }
    @Override
    @CallSuper
    public void onCarServiceStatusChange(CarServiceStatus status) {
        if (status == CarServiceStatus.CONNECTED && !mSubscribeSuccessfully) {
            subscribe();
            // backward compatible code
            if (mControllerListener != null)
                mControllerListener.onVhalConnectedW();
        }
    }
}

```

```

    }
    if (status == CarServiceStatus.DISCONNECTED) {
        mStatusSignalValueIndicator.clear();
        unsubscribe();
        mSubscribeSuccessfully = false;
        // backward compatible code
        if (mControllerListener != null)
            mControllerListener.onVhalDisconnectedW();
    }
    mPropertyManager = mLucidCar.mPropertyManager; // 断开联系后，这里的重置有可能是空值 null 吗？
}
@Override
    public void onLucidCarPropertyManagerReady(LucidCarPropertyManager lucidCarPropertyManager) {
        mPropertyManager = lucidCarPropertyManager;
    }

abstract void onChangeEvent(CarPropertyValue carPropertyValue);
// 为什么会是个链表呢？不是太不方便操作了吗？为什么不需要 Map 呢？
abstract List<Integer> getStatusSignals(); // 所胡信号标识 int 值都不同，可以是 List, Set 等

@CallSuper
    protected void cleanup() {
        mLucidCar.removeCarServiceStatusChangeListener(this);
        if (mStatusSignals != null) {
            mStatusSignals.clear();
            mStatusSignals = null;
        }
        if (mStatusSignalValueIndicator != null) {
            mStatusSignalValueIndicator.clear();
            mStatusSignalValueIndicator = null;
        }
    }
    private void subscribe() {
        mStatusSignalValueIndicator.clear();
        mStatusSignalValueIndicator.addAll(mStatusSignals);
        mLucidCar.subscribe(mStatusSignals, mCarPropertyEventCallback);
    }
    private void unsubscribe() {
        if (mStatusSignals != null) // 取消链表里所有信号的回调
            mLucidCar.unsubscribe(mStatusSignals, mCarPropertyEventCallback); //
    }
    private void checkIfNotifyStatus(int propertyId) { // 这里还是没有看懂?????......
        if (mStatusSignalValueIndicator != null && mStatusSignalValueIndicator.size() == 0) {
            if (mControllerListener != null) {
                // backward compatible code
                mControllerListener.onPropertiesReadyW();
                mControllerListener.onCarServiceStatusChangedW(CarServiceStatus.PROPERTY_VALUE_READY);
            }
        } else {
            synchronized (mLock){
                mStatusSignalValueIndicator.remove(propertyId);
            }
        }
    }
}
}

```

8.2 以 HvacController 为例，Controller LifeCycle 相关的理解

8.3 FrontSeatHvacController

```

// FrontSeatHvacController: 扩展 FrontSeatHvacListener 和覆盖重写两个回调方法
public class FrontSeatHvacController extends ControllerBase<FrontSeatHvacController.FrontSeatHvacListener>{
    public static class FrontSeatHvacListener extends ControllerListenerBase{
        public void onUpdateCushionHeatingW(int whichSeat, SeatHVACLevel cushionHeatLevel) {}
        public void onUpdateBackRestHeatingW(int whichSeat, SeatHVACLevel backRestHeatLevel) {}
        public void onUpdateCushionCoolingW(int whichSeat, SeatHVACLevel cushionCoolLevel) {}
        public void onUpdateBackRestCoolingW(int whichSeat, SeatHVACLevel backRestCoolLevel) {}
    }
    public FrontSeatHvacController() {
        mHandlerThread = new HandlerThread(TAG);
        mHandlerThread.start();
        mSignalBroadcastHandler = new SignalBroadcastHandler(mHandlerThread.getLooper());
    }
}

```

```

@Override
public void init(@NonNull Context context, FrontSeatHvacListener frontSeatHvacListener){
    mContext = context;
    mFrontSeatHvacListener = frontSeatHvacListener;
    super.init(context, frontSeatHvacListener);
}
@Override
public void init(FrontSeatHvacListener frontSeatHvacListener) { // 监听向上向 UI 回调的回调函数作为参数传进构造器
    mFrontSeatHvacListener = frontSeatHvacListener;
    super.init(frontSeatHvacListener);
}
public void cleanup() {
    if (mIsSignalDaemon)
        setSignalDaemon(false);
    super.cleanup();
}
@Override void onChangeEvent(CarPropertyValue carPropertyValue) {
    mSignalBroadcastHandler.post(new Runnable() {
        @Override public void run() {
            switch (carPropertyValue.getPropertyId()) {

                case ISEAT_LF_HEAT_STS_ZONE_2:
                case ISEAT_LF_HEAT_STS_ZONE_4:
                    // Driver Cushion Zone 2 and zone 4 values
                    if (carPropertyValue.getPropertyId() == ISEAT_LF_HEAT_STS_ZONE_2) {
                        mLeftCushionHeatZone2Value = (Integer) carPropertyValue.getValue();
                    } else if (carPropertyValue
                        .getPropertyId() == ISEAT_LF_HEAT_STS_ZONE_4) {
                        mLeftCushionHeatZone4Value = (Integer) carPropertyValue.getValue();
                    }

                    // Driver Cushion will be turned on only if both the Zone 2 and Zone 4 are equal.
                    if (mLeftCushionHeatZone2Value == mLeftCushionHeatZone4Value) {
                        handleDriverCushionHeatStateW(mLeftCushionHeatZone4Value);
                    } else {
                        Logger.d(TAG,
                            "Controller : Driver cushion Heat zone 2 and zone 4 are not equal ");
                    }
                    break;
                    // other cases
            }
        }
    });
}
@Override List<Integer> getStatusSignals() {
    return S_NOTIFY_PROP_IDS;
}
// 感觉这里的梳扭中心：现在想得比较明白一点儿了，昨天原来别人是故意不给你讲明白呀。。。。看来鱼与渔还是有很大区别的
public void setSignalDaemon(boolean isSignalDaemon) { //
    mIsSignalDaemon = isSignalDaemon;
    // 如果是 Daemon Controller: 比如 HvacApp 中的 HvacController,
    // 把自己注册成 DAEMON_INTENT_FILTER 广播信号的接收者，接收来自各 client(controller) 应用 apps 发送过来的（想要硬件调值与它们同步
    if ( mIsSignalDaemon ) {
        IntentFilter intentFilter = new IntentFilter();
        for ( String action : DAEMON_INTENT_FILTER ) {
            intentFilter.addAction( action );
        }
        Logger.d(TAG, "Daemon Instance is set to true");
        mContext.registerReceiver( mDaemonClientReceiver, intentFilter );
    } else // 是 Client Controller, 比如 Alexa 中的 HvacController?, 就取消注册任何 mDaemonClientReceiver 广播
        mContext.unregisterReceiver( mDaemonClientReceiver );
}
private static final String DAEMON_INTENT_FILTER[] = {
    ACTION_ADJUST_CUSHION_COOLING,
    ACTION_ADJUST_BACKREST_COOLING,
    ACTION_ADJUST_BACKREST_HEATING,
    ACTION_ADJUST_CUSHION_HEATING
};

```

9 BcmController

- User this Controller as the example try to understand Andorid 9 Contollers logics

9.1 BcmController.BcmListener static memeber

```
public class BcmController {
    private static final String TAG = "BcmController";
    private static final int VCU_SIGNAL_TIMEOUT = 1 * 1000; // this timeout will be used for REGEN, STOPMODE, and CREEP MOD
    //=====
    // BCM Notification Callback Listener
    //=====
    public enum CidState {
        CID_DEPLOYED,
        CID_RETRACTED,
        CID_INTERMEDIATE,
        CID_DEPLOYING,
        CID_RETRACTING,
        CID_FAULT,
    }
    private final BcmController.BcmListener mBcmListener = new BcmController.BcmListener() {
        @Override
        public void onVhalConnectedW() { }
        @Override
        public void onVhalDisconnectedW() { }
        // @Override // 现在不再能够收到这个信号了
        // public void onEngagedHvacBtnsW(boolean bEngaged) { // Soumya: 车上四个物理键被按的回调
        //     Log.d(TAG, "onhvacbtn clicked" + bEngaged); // 这个从来不曾打印, 也就是 vhal 的信号没能传上来! 就算是曾经印过,
        //     mBtnEngaged = bEngaged;
        //     if (mShowWindow)
        //         updateAutoCloseTimer();
        //     mHandler.post(mHvacBtnsRunnable); // showWindow()
        // }
        @Override public void onUpdateVolumeRollerW(int adjustValue) { // 一个特定键被按的回调
            if (mShowWindow) // 这里也就处理了两种 PeekWindow 交互的一个方向: 当正在显示, 按了音量, 就隐藏, 可能小细节还需要再处理
                hideWindow();
        }
        @Override public void onUpdateHvacBtnDriverTemperature(int state) { // 这两个方法最开始没有想明白
            mBtnEngaged = true;
            mHandler.post(mHvacBtnsRunnable);
            // 必须先开风扇, 再调温度之类的... 这里不向 UI 回调, 更像是向下去同 Bcm 同步: 如果调了温度而没有开风扇, 不显示, 但是会把车的
            if ( state >= 1 && state <= 4 && mHvacController != null &&
                mHvacController.getFanSpeed( HvacController.ZONE_FRONT_LEFT ) == 0 ) {
                int mappedFanSpeed = mapFanSpeed( 0, 10, 0, 100, 1 );
                mHvacController.setFanSpeed( HvacController.ZONE_FRONT_LEFT, mappedFanSpeed );
            }
        }
        @Override public void onUpdateHvacBtnPassengerTemperature(int state) {
            mBtnEngaged = true;
            mHandler.post(mHvacBtnsRunnable);
            if ( state >= 1 && state <= 4 && mHvacController != null &&
                mHvacController.getFanSpeed( HvacController.ZONE_FRONT_RIGHT ) == 0 ) {
                int mappedFanSpeed = mapFanSpeed( 0, 10, 0, 100, 1 );
                mHvacController.setFanSpeed( HvacController.ZONE_FRONT_RIGHT, mappedFanSpeed );
            }
        }
        @Override public void onUpdateHvacBtnDriverFanSpeed(int state) {
            mBtnEngaged = true;
            mHandler.post(mHvacBtnsRunnable);
        }
        @Override public void onUpdateHvacBtnPassengerFanSpeed(int state) {
            mBtnEngaged = true;
            mHandler.post(mHvacBtnsRunnable);
        }
    }
}
```

9.2 Connection to VHAL

```
//=====
// Init / Cleanup
//=====
private Context mContext;
private BcmListener mBcmListener;
public BcmController() {
    // Signal broadcasting worker thread.
    mHandlerThread = new HandlerThread( TAG );
```

```

        mHandlerThread.start();
        mSignalBroadcastHandler = new SignalBroadcastHandler( mHandlerThread.getLooper() );
    }
    public void init(Context context, BcmListener bcmListener) {
        mContext = context;
        mBcmListener = bcmListener;
        mCarApiClient = Car.createCar( mContext, mCarConnectionCallback );
        mCarApiClient.connect(); // 先连车
    }
    public void cleanup() {
        if ( mCarApiClient != null )
            mCarApiClient.disconnect();
        mBcmListener = null;
        mContext = null;
    }
    //=====
    // Connection to VHal
    //=====
    private Car mCarApiClient;
    private CarPropertyManager mPropertyManager;
    private final ServiceConnection mCarConnectionCallback = new ServiceConnection() {
        @SuppressWarnings("StaticFieldLeak")
        @Override
        public void onServiceConnected(ComponentName name, IBinder service) {
            if ( mBcmListener != null ) {
                if ( IS_DEBUG_BUILD ) Log.d( TAG, "mBcmListener.onVhalConnectedW()" );
                mBcmListener.onVhalConnectedW(); // 调用应用层它们的实现，它们还需要或者可以额外的做点什么事儿呢？
            }
            try { // 获取 requestSignal 初始化数据
                mPropertyManager = (CarPropertyManager) mCarApiClient
                    .getCarManager( android.car.Car.PROPERTY_SERVICE );
                for ( int propId : S_NOTIFY_PROP_IDS ) // 为当前控制器所监控的每个属性，注册回调，rate=100
                    // 为某个车的属性注册回调：注册的过程本身就一定会收到（初始化的）回调
                    mPropertyManager.registerCallback(mPropertiesListener, propId, 100 );
                // 这里，上面，当第一次注册的时候，总是会返回 true（？不是，第一次注册的时候，很多地方会返回 false）
                // Fetch initial values. 异步执行
                new AsyncTask<Void, Void, Void>() { // 这个异步：只是更新了车 CarPropertyManager 里的成员变量的值，并不曾回调出什么
                    @Override protected Void doInBackground(Void... nulls) {
                        fetchProperties(); // 它并没有处理 INPUTS_HVAC_BTNS_ENGAGED 这一属性（2种初始化方式：fetch or registerC
                        return null;
                    }
                }
                @Override protected void onPostExecute(Void aVoid) { // 那么这里的初始化，关这一属性什么事儿呢？
                    // Notify
                    if ( mBcmListener != null ) {
                        if ( IS_DEBUG_BUILD ) Log.d( TAG, "mBcmListener.onPropertiesReadyW()" );
                        mBcmListener.onPropertiesReadyW(); // 当异步任务执行完了之后的回调，对于那条属性来说，不是也没有处理吗？
                    }
                    // 我觉得上面这一行，在 VHAL 或者某个 层面，应该有一个自动化填充某些值的过程
                }
            }
            }.execute();
        } catch ( CarNotConnectedException e ) {
            Log.e(TAG, "onConnected() : Car not connected.");
        } catch ( IllegalStateException e ) {
            Log.e(TAG, "Illegal State Exception... " + e.getMessage());
        }
    }
    @Override public void onServiceDisconnected(ComponentName name) {
        for ( int propId : S_NOTIFY_PROP_IDS )
            mPropertyManager.unregisterCallback(mPropertiesListener, propId );
        if ( mBcmListener != null ) {
            if ( IS_DEBUG_BUILD ) Log.d( TAG, "mBcmListener.onVhalDisconnectedW()" );
            mBcmListener.onVhalDisconnectedW();
        }
    }
}
};

```

9.3 Incoming BCM Signals Handling

```

//=====
// Incoming BCM Signals Handling
//=====
private CarPropertyManager.CarPropertyEventCallback mPropertiesListener = // 车的任何属性改变事件的回调：出错了，或是状态变化了
    new CarPropertyManager.CarPropertyEventCallback() {

```

```

@Override
public void onErrorEvent(int i, int i1) {
    Log.e( TAG, "onErrorEvent() : i=" + i + ", i1=" + i1 );
}
@Override
public void onChangeEvent(CarPropertyValue carPropertyValue) {
    if (IS_DEBUG_BUILD)
        Log.d(TAG, " onChangeEvent " + carPropertyValue.getPropertyId()
            + " (0x" + Integer.toHexString(carPropertyValue.getPropertyId()) + ")");
    if (carPropertyValue.getPropertyId() == INPUTS_HVAC_BTNS_ENGAGED) {
        Log.d(TAG, "(carPropertyValue.getPropertyId() == INPUTS_HVAC_BTNS_ENGAGED): " + (carPropertyValue.getPropertyId()
            Log.d(TAG, "(Boolean)carPropertyValue.getValue(): " + (Boolean)carPropertyValue.getValue());
    }
    switch ( carPropertyValue.getPropertyId() ) {
    case CID_POSITION_STATUS :
        handleUpdateCidStateW( (Integer) carPropertyValue.getValue() );
        break;
// 这个东西没有了
// case INPUTS_HVAC_BTNS_ENGAGED : // 如果时间足够短, 也不会传到 onChangeEvent(CarPropertyValue carPropertyValue)
//     handleUpdateHvacBtnsEngagedW( (Boolean) carPropertyValue.getValue() );
//     break;
    case INPUTS_VOLUME_ROLLER : // 监听 Roller 被按的情况
        handleUpdateVolumeRollerW( (Integer) carPropertyValue.getValue() );
        break;
    case INPUTS_HVAC_BTN_DRIVER_TEMPERATURE: // 两个调温按钮
        handleUpdateHvacBtnDriverTemperature( (Integer) carPropertyValue.getValue() );
        break;
    case INPUTS_HVAC_BTN_PASSENGER_TEMPERATURE:
        handleUpdateHvacBtnPassengerTemperature( (Integer) carPropertyValue.getValue() );
        break;
// 我们现在需要监听风扇的情况, 需要与 HvacController 对比一下, 实现起来有什么不同, 为什么后者分不出是否有物理键
// 这里传回来的都是实际的温度, 我怎样才能让它付出 boolean 的值呢?
    case INPUTS_HVAC_BTN_DRIVER_FANSPEED:
        handleUpdateHvacBtnDriverFanSpeed( (Integer) carPropertyValue.getValue() );
        break;
    case INPUTS_HVAC_BTN_PASSENGER_FANSPEED: // 我们现在需要监听风扇的情况, 需要与 HvacController 对比一下, 实现起来有什么不同
        handleUpdateHvacBtnPassengerFanSpeed( (Integer) carPropertyValue.getValue() );
        break;
    // .....
    }
}
void handleUpdateHvacBtnDriverTemperature(Integer value) {
    if ( mBcmListener != null ) {
        if (IS_DEBUG_BUILD) Log.d( TAG, "mBcmListener.onUpdateHvacBtnDriverTemperature() : value=" + value );
        mBcmListener.onUpdateHvacBtnDriverTemperature( value );
    }
}
// 这里需要自己定义, 那么也同样需要添加 BcmListener 所定义的接口; 或者去翻 HvacController.java 参考一下
// 那么对应的, 上面的 BcmController.BcmListener 的回调接口也需要添加, 见上
void handleUpdateHvacBtnDriverFanSpeed(Integer val) {
    if ( mBcmListener != null ) {
        if (IS_DEBUG_BUILD) Log.d( TAG, "mBcmListener.onUpdateHvacBtnDriverFanSpeed() : value=" + value );
        mBcmListener.onUpdateHvacBtnDriverFanSpeed( value );
    }
}
void handleUpdateHvacBtnPassengerFanSpeed(Integer value) {
    if ( mBcmListener != null ) {
        if (IS_DEBUG_BUILD) Log.d( TAG, "mBcmListener.onUpdateHvacBtnPassengerFanSpeed() : value=" + value );
        mBcmListener.onUpdateHvacBtnPassengerFanSpeed( value );
    }
}
}
// framework: 为某个车的属性注册回调, 注册的过程, 注册的这一次本身就一定会收到回调, 这里需要再好好想一想.....
private final SparseArray<CarPropertyManager.CarPropertyListeners> mActivePropertyListener = new SparseArray();
public boolean registerCallback(@NonNull CarPropertyEventCallback callback,
    int propertyId, @FloatRange(from = 0.0, to = 100.0) float rate) {
    synchronized (mActivePropertyListener) {
        if (mCarPropertyEventToService == null)
            mCarPropertyEventToService = new CarPropertyEventListenerToService(this);
        CarPropertyConfig config = mConfigMap.get(propertyId); // 这里说了, 这是一个字典 [key, value] Pair
        if (config == null) { // 如果字典本身就没有这属性, 返回假
            Log.e(TAG, "registerListener: propid is not in config list: " + propertyId);
            return false;
        }
        if (config.getChangeMode() == CarPropertyConfig.VEHICLE_PROPERTY_CHANGE_MODE_ONCHANGE)

```

```

        rate = SENSOR_RATE_ONCHANGE; // 对于一辆车来说，它的某些属性的改变事件有它自己的定义，并不真正意味着任何属性都有一个
        boolean needsServerUpdate = false; // 需要更新属性，通知监听都吗？
        CarPropertyListeners listeners;
        listeners = mActivePropertyListener.get(propertyId); // 某个属性的所有监听者回调，有点儿像字典，但不是字典
        if (listeners == null) { // 如果这个属性暂时没有监听者，就为它注册一个？
            listeners = new CarPropertyListeners(rate); // listeners = new ArrayList<>(); ??? 是这样的吗？
            mActivePropertyListener.put(propertyId, listeners);
            needsServerUpdate = true;
        }
        if (listeners.addAndUpdateRate(callback, rate)) // 对于当前回调，现链表里的元素，需要添加或是更新吗？
            needsServerUpdate = true;
        if (needsServerUpdate) {
            if (!registerOrUpdatePropertyListener(propertyId, rate)) { // 对于当前车属性，需要添加或是更新回调链表里的内容吗？
                return false;
            }
        }
    }
    return true;
}
// 这个数据结构的用法，还需要再熟悉一下
private final SparseArray<CarPropertyManager.CarPropertyListeners> mActivePropertyListener = new SparseArray();
// 这里关于
private class CarPropertyListeners extends CarRatedFloatListeners<CarPropertyManager.CarPropertyEventCallback> {
    CarPropertyListeners(float rate) {
        super(rate);
    }
    void onPropertyChanged(final CarPropertyEvent event) {
        final long updateTime = event.getCarPropertyValue().getTimestamp();
        int areaId = event.getCarPropertyValue().getAreaId();
        if (!this.needUpdateForAreaId(areaId, updateTime)) {
            Log.w("CarPropertyManager", "dropping old property data");
        } else {
            ArrayList listeners;
            synchronized (CarPropertyManager.this.mActivePropertyListener) {
                listeners = new ArrayList(this.getListeners());
            }
            listeners.forEach (new Consumer<CarPropertyManager.CarPropertyEventCallback>() {
                public void accept(CarPropertyManager.CarPropertyEventCallback listener) {
                    if (CarPropertyListeners.this.needUpdateForSelectedListener(listener, updateTime))
                        listener.onChangeEvent(event.getCarPropertyValue());
                }
            });
        }
    }
    void onErrorEvent(CarPropertyEvent event) {
        final CarPropertyValue value = event.getCarPropertyValue();
        ArrayList listeners;
        synchronized(CarPropertyManager.this.mActivePropertyListener) {
            listeners = new ArrayList(this.getListeners());
        }
        listeners.forEach(new Consumer<CarPropertyManager.CarPropertyEventCallback>() {
            public void accept(CarPropertyManager.CarPropertyEventCallback listener) {
                listener.onErrorEvent(value.getPropertyId(), value.getAreaId());
            }
        });
    }
}
}

```

9.4 LucidSysUiService.java 中初始化的顺序

```

@Override public void onCreate() {
    super.onCreate();
    ^^ILog.d(TAG, "onCreate Called");
    ensureForeground(); // Make foreground service.

    mHandler = new Handler();
    context = getBaseContext();
    mLocalBroadcastManager = LocalBroadcastManager.getInstance(this.getApplicationContext());

    // IcrSettingsWindow...
    createAlarmManager(getApplicationContext(), false);
    mLucidStatusBarWindow = new LucidStatusBarWindow(context);
    mIcrNavigationBarWindow = new IcrNavigationBarWindow(context);
    mIcrNavigationBarWindow.showWindow();
}

```

```

mLucidStatusBarWindow.showStatusBar();

// HVAC & volume inputs feedback window...
mIcrHvacPeekWindow = new IcrHvacPeekWindow(context); // 1
mIcrSettingsWindow = new IcrSettingsWindow(context);
mCleanScreenWindow = new CleanScreenWindow(context);
mOtaNotificationWindow = new OtaNotificationWindow(context);
otaIntent = new Intent();
seatsNotificationWindow = new SeatsNotificationWindow(context);
mBugReportNotificationWindow = new BugReportNotificationWindow(context);
mFteNotification = new FteNotification(context);
//init TcuController
mTcuController = new TcuController(); // 2
mTcuController.init(getApplicationContext(), mTcuVHALConnectionListener, 1000, mTcuPropertiesListener);
mIcrVolumeWindow = new IcrVolumeWindow(getApplicationContext());

int level = UserProfileData.retrieveIntData(ProfileKey.SETTINGS_ADAS_LDP_LEVEL,
                                           AdasController.LDP_LEVEL_INTERVENTION);

if (mAdasController == null) mAdasController = new AdasController();
Context context = getApplicationContext();
mAdasController.init(context, new AdasController.AdasListener() {
    @Override
        public void onPDWTriggered(boolean isPdw) {
            if (isPdw) {
                LucidSystemUiUtil.sendCidNavBarCommand(context, CID_NAV_BAR_COMMAND_DEPLOY_CID);
            }
        }, level);
}
}

```

- log 的总体纪录：这里可以清楚地看见 `IcrHvacPeekWinodw` 与 `IcrVolumeWindow` 每个分别把 `BcmController.BcmListener` 给初始化了一遍，如果一个应用有 1000 个倾听者，会怎么样呢？
- 暂且就当自己是明白了吧，先去把自己的活儿干完

```
<<<<<<<<===== IcrHvacPeekWindow() constructor 中四个 Controller.Listener.init() 的依次回调
IcrHvacPeekWindow: Park Assist Controller connected <<<<<<<<===== IcrHvacPeekWindow() constr
IcrHvacPeekWindow: Gear Controller VHAL connected
IcrHvacPeekWindow: Bcm Controller VHAL connected
BcmController: fetchProperties(): BEGIN
BcmController: ONE_PEDAL_DRIVING_STS > ( 0 )
BcmController: fetchProperties(): END
IcrHvacPeekWindow: Hvac Vhal connected

LucidSysUIService: TCU vHal connected
IcrVolumeWindow: Bcm Controller VHAL connected <<<<<<<<===== 这里 IcrVolumeWindow 在这里又联了
TcuController: fetchWifiSignalState strength: 2 status: 2 mCtrlsListener: com.lucid.car.sysui.LucidSysUIService$5@a9b2962
LucidSysUIService: onWifiConnectionStatusChangedW wifiStatus: 2
LucidSysUIService: onWifiSignalUpdate Wifi_Strength:..... 2
TcuController: fetchLteSignalState strength: 3 status: 2 mCtrlsListener: com.lucid.car.sysui.LucidSysUIService$5@a9b2962
LucidSysUIService: onLteConnectionStatusChangedW Lte_Status: 2 Wifi_Status: 2
BcmController: fetchProperties(): BEGIN <<<<<<<<===== 这里 IcrVolumeWindow 在这里又联了一遍，
IcrVolumeWindow: SteeringWheel Controller VHAL connected
BcmController: ONE_PEDAL_DRIVING_STS > ( 0 )
BcmController: fetchProperties(): END

IcrVolumeWindow: restoreVolumeLevel()
```

```

IcrVolumeWindow: restoreVolumeLevel(): groupId=0, vol=10
IcrVolumeWindow: onGroupVolumeChanged()
IcrVolumeWindow: ( flags & AudioManager.FLAG_SHOW_UI ): 0
IcrVolumeWindow: muteUnmute() groupId: 0 volume: 10
IcrVolumeWindow: LUCID_MEDIA_MUTE=0
IcrVolumeWindow: persistVolumeLevel(): groupId=0, vol=10
IcrVolumeWindow: restoreVolumeLevel()
IcrVolumeWindow: restoreVolumeLevel(): groupId=1, vol=17
IcrVolumeWindow: onGroupVolumeChanged()
IcrVolumeWindow: ( flags & AudioManager.FLAG_SHOW_UI ): 0
IcrVolumeWindow: muteUnmute() groupId: 1 volume: 17
IcrVolumeWindow: LUCID_NAVIGATION_MUTE=0
IcrVolumeWindow: persistVolumeLevel(): groupId=1, vol=17
IcrVolumeWindow: restoreVolumeLevel()
IcrVolumeWindow: restoreVolumeLevel(): groupId=2, vol=17
IcrVolumeWindow: onGroupVolumeChanged()
IcrVolumeWindow: ( flags & AudioManager.FLAG_SHOW_UI ): 0
IcrVolumeWindow: muteUnmute() groupId: 2 volume: 17
IcrVolumeWindow: LUCID_PHONE_MUTE=0
IcrVolumeWindow: persistVolumeLevel(): groupId=2, vol=17

LucidSysUiService: Steering Wheel VHAL Connected
LucidSysUiService: Steering Wheel property ready

IcrHvacPeekWindow: CCCQNXAppStatus value = -1 来自 IcrHvacPeekWindow 的 ParkAssistantController
IcrHvacPeekWindow: Gear value = 4
IcrHvacPeekWindow: Gear value = 4

BcmController: (carPropertyValue.getPropertyId() == INPUTS_HVAC_BTNS_ENGAGED): true
BcmController: (Boolean)carPropertyValue.getValue(): false
IcrHvacPeekWindow: onhvacbtn clickedfalse
BcmController: handleOnUpdateOnePedalDrivingW 0
BcmController: reported regen braking level is invalid
BcmController: IVCU_LSC_MODE_STAT value 0
BcmController: LSC Mode update value 0
BcmController: creep mode 0
BcmController: creep mode status = 0
IcrHvacPeekWindow: fan speed changed2317
IcrHvacPeekWindow: fan speed changed2318
IcrHvacPeekWindow: fan speed changed033
IcrHvacPeekWindow: fan speed changed034
IcrHvacPeekWindow: temp changed15017
IcrHvacPeekWindow: temp changed16718
IcrHvacPeekWindow: temp changed15033
IcrHvacPeekWindow: temp changed15034
LucidSysUiService: onWifiSignalUpdate Wifi_Strength:..... 2
LucidSysUiService: onWifiConnectionStatusChangedW wifiStatus: 2
LucidSysUiService: onLteConnectionStatusChangedW Lte_Status: 2 Wifi_Status: 2
LucidStatusBarView: onReceive of mLucidSysUiReceiver: com.lucid.car.settings.LucidSysUiService.ACTION_WIFI_SIGNAL
LucidStatusBarView: onReceive of mLucidSysUiReceiver: com.lucid.car.settings.LucidSysUiService.ACTION_WIFI_SIGNAL
LucidStatusBarView: onReceive of mLucidSysUiReceiver: com.lucid.car.settings.LucidSysUiService.ACTION_WIFI_SIGNAL
BcmController: (carPropertyValue.getPropertyId() == INPUTS_HVAC_BTNS_ENGAGED): true
BcmController: (Boolean)carPropertyValue.getValue(): false
IcrVolumeWindow: onhvacbtn clickedfalse
IcrVolumeWindow: scroll number from vhal: 1
IcrVolumeWindow: adjustVolume(): adjustValue=1
IcrVolumeWindow: adjustVolume(): groupId=0, vol=11
IcrVolumeWindow: onGroupVolumeChanged()
IcrVolumeWindow: ( flags & AudioManager.FLAG_SHOW_UI ): 1
IcrVolumeWindow: mVolGroupId: 0
IcrVolumeWindow: muteUnmute() groupId: 0 volume: 11
IcrVolumeWindow: persistVolumeLevel(): groupId=0, vol=11
IcrVolumeWindow: LUCID_MEDIA_MUTE=0
BcmController: handleOnUpdateOnePedalDrivingW 0
BcmController: reported regen braking level is invalid
BcmController: IVCU_LSC_MODE_STAT value 0
BcmController: LSC Mode update value 0
BcmController: creep mode 0
BcmController: creep mode status = 0
IcrVolumeWindow: onUpdateRockerRightButtonW()
IcrVolumeWindow: onUpdateToggleRightW()

```

10 VHAL TEAM 说他们现在不传这个信号给我们了，我们需要自己来合并

10.1 涉及到的物理键及相关链接

- 下面是 BcmController 四个物理键的回调：下面我贴的不一定正确，真正的来源是在<https://lucidmotors.atlassian.net/wiki/spaces/INF0/pages/1369538838/CCC+Signals>

Feature	Request Signal	Status Signal	Values
Fan speed physical button (driver & passenger)		HVAC_Status/IHVAC_Key2Sts	Not_Pressed:0, Up_Single_Press:1, Up_Press_Hold:2, Down_Single_Press:3
		HVAC_Status/IHVAC_Key3Sts	Down_Press_Hold:4, TBD:5, Failure_Detected:6, Invalid:7
Temperature physical button (driver & passenger)		HVAC_Status/IHVAC_Key1Sts	Not_Pressed:0, Up_Single_Press:1, Up_Press_Hold:2, Down_Single_Press:3
		HVAC_Status/IHVAC_Key4Sts	Down_Press_Hold:4, TBD:5, Failure_Detected:6, Invalid:7

- LucidVehSigUtil.h

```
INPUTS_HVAC_BTN_DRIVER_TEMPERATURE,
INPUTS_HVAC_BTN_DRIVER_FANSPEED,
INPUTS_HVAC_BTN_PASSENGER_FANSPEED,
INPUTS_HVAC_BTN_PASSENGER_TEMPERATURE,

const string kPduHvacStatus = "HVAC_Status";
const string kHvacKeyPrefix = "IHVAC_Key";

const string kHvacScrollPrefix = "IHVAC_Scroll";
const string kSigHvacScrollDirection = "IHVAC_ScrolllDirection";
const string kSigHvacScrollsNumber = "IHVAC_ScrollsNumber";

const string kSigHvacKey1Sts = "IHVAC_Key1Sts"; // Driver Temp
const string kSigHvacKey2Sts = "IHVAC_Key2Sts"; // Driver Fan
const string kSigHvacKey3Sts = "IHVAC_Key3Sts"; // Passenger Fan
const string kSigHvacKey4Sts = "IHVAC_Key4Sts"; // Pasenger Temp
```

10.2 VHAL 层面 HvacPropertyHelper.cpp 的实现逻辑

```
bool get_hvac_hw_switch_prop(SignalPathTuple signal_path, double value,
                             VehiclePropValuePool* value_pool,
                             std::vector<VehicleHal::VehiclePropValuePtr& prop_vals) {

    bool handled = false;
    std::string sig_name = std::get<2>(signal_path);
    auto val_prop = value_pool->obtain(VehiclePropertyType::INT32);
    val_prop->timestamp = elapsedRealtimeNano();
    val_prop->areaId = 0;
    // Both of the following is for the volume scroller
    if (sig_name.find(kHvacScrollPrefix) != std::string::npos) {
        if (sig_name == kSigHvacScrollDirection) {
            mRollerState[RollerStateIndex::SCROLL_DIR] = (int) value; // 0, 1, or 2
        }
        if (sig_name == kSigHvacScrollsNumber) {
            mRollerState[RollerStateIndex::SCROLL_VAL] = (int) value;
            // invalidate non-zero value so vhal can be notified again even if
            // value hasn't changed
            if (0 != value) {
                invalidate_signal(signal_path);
            }
        }
    }
    bool roll_dir_valid = (mRollerState[RollerStateIndex::SCROLL_DIR] != 0 &&
                           mRollerState[RollerStateIndex::SCROLL_DIR] != 3);
```

```

    if (roll_dir_valid && mRollerState[RollerStateIndex::SCROLL_VAL] != 0) {
        handled = true;
        int dir = (mRollerState[RollerStateIndex::SCROLL_DIR] == RollerDirection::UP) ? 1 : -1;
        int vol = dir * mRollerState[RollerStateIndex::SCROLL_VAL];
        if (is_debuggable) {
            ALOGD("%s: Roller engaged: %d", __FUNCTION__, vol);
        }
        val_prop->prop = LucidVehicleProperty::INPUTS_VOLUME_ROLLER;
        val_prop->value.int32Values[0] = vol;
    }
    if (handled) {
        prop_vals.push_back(std::move(val_prop));
    }
    // Handle hvac hard btns specifically ?????// 是这个吗???
} else if (sig_name.find(kHvacKeyPrefix) != std::string::npos) {
    handled = handle_hvac_hard_buttons(signal_path, value, value_pool, prop_vals);
}
return handled;
}

// 这个应该就是相关的最重要的方法了：怎么还是没有看懂呢????
bool handle_hvac_hard_buttons(SignalPathTuple signal_path,
                             double value,
                             VehiclePropValuePool *value_pool,
                             std::vector<VehicleHal::VehiclePropValuePtr> &prop_vals) {
    const auto &sig_name = std::get<2>(signal_path);
    if (sig_name == kSigHvacKey2Sts) {
        mHvacBtnState[PhysicalHwButton::DRV_FAN] = (int) value;
    }
    if (sig_name == kSigHvacKey1Sts) {
        mHvacBtnState[PhysicalHwButton::DRV_TEMP] = (int) value;
    }
    if (sig_name == kSigHvacKey3Sts) {
        mHvacBtnState[PhysicalHwButton::PASS_FAN] = (int) value;
    }
    if (sig_name == kSigHvacKey4Sts) {
        mHvacBtnState[PhysicalHwButton::PASS_TEMP] = (int) value;
    }
    int btn_state = 0;
    for (int state : mHvacBtnState) {
        if (state > 0 && state < 5) {
            btn_state = 1;
            if (is_debuggable) {
                ALOGD("%s: HVAC h/w button engaged", __FUNCTION__);
            }
            break;
        }
    }
    auto val_prop = value_pool->obtain(VehiclePropertyType::INT32);
    val_prop->timestamp = elapsedRealtimeNano();
    val_prop->areaId = 0;
    val_prop->prop = LucidVehicleProperty::INPUTS_HVAC_BTNS_ENGAGED;
    val_prop->value.int32Values[0] = btn_state; // 这里不是设置了一遍吗？
    prop_vals.push_back(std::move(val_prop));

    auto hvac_btn_prop = value_pool->obtain(VehiclePropertyType::INT32);
    hvac_btn_prop->timestamp = elapsedRealtimeNano(); // 要这个是干什么用的 ??? 超时，过了某个界限，视为 release 没有再按
    hvac_btn_prop->areaId = 0;
    hvac_btn_prop->prop = k_vsigt_rx_to_prop_id_map[LucidVehSigUtil::sig_tuple_to_str(signal_path)];
    hvac_btn_prop->value.int32Values[0] = static_cast<int>(value); //
    prop_vals.push_back(std::move(hvac_btn_prop));
    return true;
}
}

```

- 能否找点日志出来看一下这些是什么呢？