Lucid Air 安卓高端个性化配置 OS – 系统配置启动过程：我想要看见更高的天空!!!

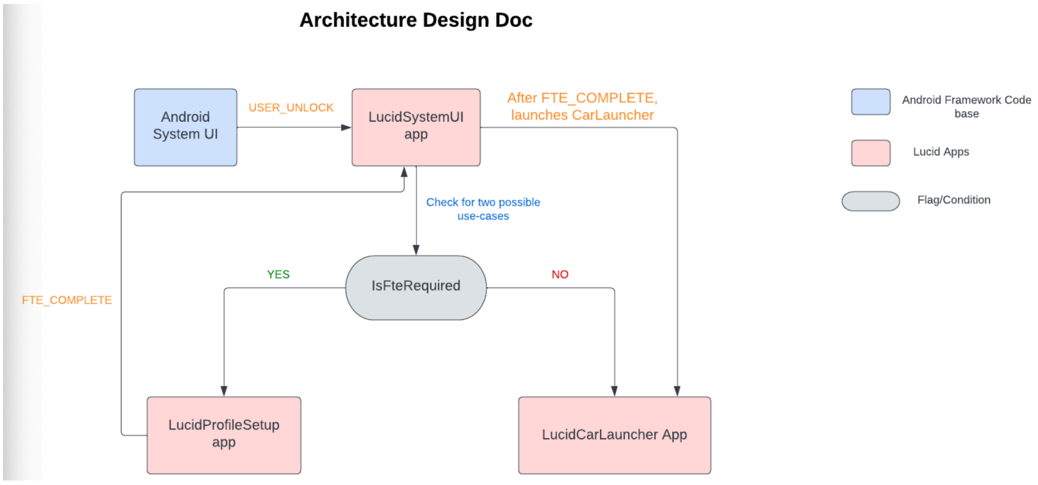
deepwaterooo

May 21, 2022

Contents

1 LucidSystemUI LucidCarLauncher LucidProfileSetup apps launching process: 程序的启动逻辑可以很灵活地随时改动，这里未必是当前最新版本 LucidSystemUI

- https://lucidmotors.atlassian.net/wiki/spaces/INFO/pages/2730492831/Architecture+design+proposal+for+bootup+sequence+apps
- proposes a new architecture design for having a new centralized place to control the LucidSystemUI app, LucidCarLauncher app & LucidProfileSetup app for various boot-up sequence use-cases as well as new profile creation use-cases.
- adding diagrams for all apps, adding sequences, listing out all limitations of current architecture, and introducing advantages with a new architecture design.



1.1 BOOT: BOOT_COMPLETE, BOOT_COMPLETED 相关, Lucid Air boot 的过程理解

- 这里最好是与 hvac_bug.org 中 log 抓出来的日志结合起来，去帮助自己理解这个私人定制的个性化安卓系统的启动过程，多用户切换以及电源管理等比较好用的安卓 10 特性。
- 接听 BOOT_COMPLETED 广播事件的几个应用与服务及其先后顺序:
- 就包括了 LucidSystemService, LucidDmsService, com.lucid.car.carsettings.BootReceiver, InjectBaseVersionS2110, com.lucid.car.sysui.pinToDrive.PinToDriveService

```
I AlarmClock: AlarmInitReceiver android.intent.action.LOCKED_BOOT_COMPLETED
I ActivityManager: Posting BOOT_COMPLETED user #10
D LucidSystemService: onReceive() ACTION_USER_UNLOCKED
D LucidSystemService: onReceive: intent android.intent.action.USER_UNLOCKED
D LucidSystemService: (intent.hasExtra(EXTRA_USER_HANDLE)): true
D LucidSystemService: START LucidCarLauncher
D LucidSystemService: START LucidSysUiService.java
D LucidSystemService: onStartCommand    :isFteRequiredfalse
D ONSAutoboot: Received android.intent.action.LOCKED_BOOT_COMPLETED
D LucidDmsService: onReceive(): android.intent.action.BOOT_COMPLETED
D AlertReceiver: onReceive: a=android.intent.action.BOOT_COMPLETED Intent { act=android.intent.action.BOOT_COMPLETED flg=0x
I AlarmClock: AlarmInitReceiver android.intent.action.BOOT_COMPLETED
D MapApplication: received the boot complete event
E com.lucid.car.carsettings.BootReceiver: BOOT COMPLETED CALLED FROM CAR SETTINGS
D InjectBaseVersionS2110: onReceive() : intent=android.intent.action.BOOT_COMPLETED // corner case 处理: 只是给 OTA 监听者传
D com.lucid.car.sysui.pinToDrive.PinToDriveService: onBoot completed...
D LucidSysUiService: Boot up completedandroid.intent.action.BOOT_COMPLETED
D LucidSysUiService: creating alarm manager service
I ActivityManager: Finished processing BOOT_COMPLETED for u10
D LucidSystemService: onReceive() ACTION_USER_UNLOCKED
D LucidSystemService: onReceive: intent android.intent.action.USER_STOPPED
```

- 这里需要强调: LucidUserProfile 应该是广播事件的最早接收者 (优先级 3000)，只是不知道与 LucidCarService 相比，哪个先哪个后呢? 后者是在系统用户下就启动了的/???

1.2 用户的切进切出 ACTION_USER_XXXX related: current user MESSAGE/ACTION_USER_UNLOCKED, then prev user (ACTION_)USER_STOP 用户的切进切出 profile switch 系统相关事件 mark

```
D LucidSystemService: onReceive() ACTION_USER_UNLOCKED
D LucidSystemService: onReceive: intent android.intent.action.USER_UNLOCKED
D LucidSystemService: (intent.hasExtra(EXTRA_USER_HANDLE)): true
D LucidSystemService: START LucidCarLauncher <<<<<<<<<<<<<<<<<<=======================
D LucidDmsService: On received HMI action: android.intent.action.USER_UNLOCKED
D LucidSystemService: START LucidSysUiService.java <<<<<<<<<<<<<<<<<=======================
D LucidSystemService: onStartCommand    :isFteRequiredfalse
D BluetoothManagerService: MESSAGE_USER_UNLOCKED <<<<<<<<<<<<<<<<<=======================
D LucidDmsService: On received HMI action: android.intent.action.USER_STOPPED 应该是前用户停止
E ActivityManager: Sending non-protected broadcast com.lucid.home.ProfileSwitcher.ACTION_USER_LOGGED_IN from system 12202:c
D LucidDmsService: On received HMI action: android.intent.action.USER_STOPPED
D LucidSystemService: onReceive() ACTION_USER_UNLOCKED   现用户完成 <<<<<<<<<<<<<<<<<=======================
D LucidSystemService: onReceive: intent android.intent.action.USER_STOPPED 前用户停止 <<<<<<<<<<<<<<<<<=====================
```

1.3 Lucid Air 一些应用/服务的启动先后顺序关系，以及与多用户切换相关的进程控制

- 这里需要注意,在目前的多用户系统下,lucid air 所有的应用里,只有这个 com.lucid.car.service 会同时存在多个 process （ps -A | grep -i service）。
- 这里需要注意，AIDL 的 ProfileData 和 com.lucid.userprofile 如同 headless system user0，为所有用户所共同享用，唯一实例运行在 system user 0 下 (不知道是不是只在这个下，还是说它同 LucidCarService 一样，为其它非系统用户的用户数据创建与切换提供了系统级别的数据库 IPC AIDL 进程间通信的大后台？)
- LucidSysUiServices。java 等都只存在一个前台服务，切换用户的时候，目前的做法仍然是把前用户的所有相关全部杀死。

1.3.1 LucidCarService app：multiple instance，one for each user

```
px3lucid:/ # ps -A | grep -i com.lucid.car.service
system        2699   839 5039924  56876 SyS_epoll_wait      0 S com.lucid.car.service // system u0
u10_system    5047   839 5113720  58800 SyS_epoll_wait      0 S com.lucid.car.service // primary driver
# 这之间执行切换用户的操作
px3lucid:/ # ps -A | grep -i com.lucid.car.service
system        2699   839 5092244  55244 SyS_epoll_wait      0 S com.lucid.car.service // system u0: 同一进程，不曾杀死与重建
u11_system   21639   839 5176584  56608 SyS_epoll_wait      0 S com.lucid.car.service // secondary driver
```

1.3.2 Settings.Globla 一个自己不太懂的 ContentProvider 相关的东西？几个应用的相对启动顺序

```
D BluetoothManagerService: Bluetooth boot completed
D KeyguardClockSwitch: Updating clock: 606

<<<<<<<<<<============= 这里不是搞不懂吗，它在一个相对更高的层面上注册了 Settings.Global ContentProvider 看这里就够了，其它可以跳
V SettingsProvider: Notifying for 0: content://settings/global/location_global_kill_switch
V SettingsProvider: Notifying for 0: content://settings/global/location_global_kill_switch

D CarUserService: onCreate()
D VoiceInteractionManagerService: switchImplementation(10) took to complete: 1ms
D LucidCarServiceApp: onCreate...
D LucidCarServiceApp: onCreate...
D LucidCarServiceApp: startLucidSystemServices()
I SecureElementService: main onCreate
W ContextImpl: Calling a method in the system process without a qualified user: android.app.ContextImpl.startService:1570
android.content.ContextWrapper.startService:669 com.lucid.car.service.LucidCarServiceApp.startLucidSystemServices:71
com.lucid.car.service.LucidCarServiceApp.onCreate:45
android.app.Instrumentation.callApplicationOnCreate:1189
D MainActivity: onCreate
D LucidSystemService: LucidSystemService is created.
D MediaFragment: onCreateView
D LucidTimeZoneService: onCreate timeZone: America/Los_Angeles
D BluetoothAdapterService: onCreate()
I Telecom : MissedCallNotifierImpl: reloadFromDatabase: Boot not yet complete -- call log db may not be available. Deferrin
D HereNavigationNotificationService: onCreate()
W FingerprintService: User switched while performing cleanup
D MainActivity: onCreate
D MediaFragment: onCreateView
D MediaFragment: onCreateView
D CarLatinIME: onCreate
```

```
D CarLatinIME: onCreate
W ActivityManager: User switch slowed down by observer #3 KeyguardUpdateMonitor: result sent after 687 ms
D ActivityManager: Continue user switch oldUser #0, newUser #10
W BroadcastQueue: Skipping deliver [background] BroadcastRecord{898e2e2 u0 android.intent.action.BOOT_COMPLETED} to Receive
E AndroidRuntime: ^^Iat com.android.service.ims.presence.EABService.onStartCommand(EABService.java:312)
E AndroidRuntime: ^^Iat com.android.service.ims.presence.EABService.onStartCommand(EABService.java:312)
E AndroidRuntime: ^^Iat com.android.service.ims.presence.EABService.onStartCommand(EABService.java:312)
D KeyguardClockSwitch: Updating clock: 606
D AlertReceiver: onReceive: a=android.intent.action.BOOT_COMPLETED Intent { act=android.intent.action.BOOT_COMPLETED flg=0x
I AlarmClock: AlarmInitReceiver android.intent.action.BOOT_COMPLETED
D HfpClientConnService: onCreate
D HfpClientConnService: onStartCommand Intent { cmp=com.android.bluetooth/.hfpclient.connserv.HfpClientConnectionService }
D MainActivity: onCreate
D MediaFragment: onCreateView
D HereNavigationNotificationService: onCreate()
D PhoneSwitcherNetworkRequstListener: Registering NetworkFactory
D ConnectivityService: Got NetworkFactory Messenger for PhoneSwitcherNetworkRequstListener
I VvmSimStateTracker: android.intent.action.BOOT_COMPLETED
D PhoneSwitcherNetworkRequstListener: got request NetworkRequest [ TRACK_DEFAULT id=15, [ Capabilities: INTERNET&NOT_RESTRI
D PhoneSwitcherNetworkRequstListener: got request NetworkRequest [ TRACK_DEFAULT id=12, [ Capabilities: INTERNET&NOT_RESTRI
D PhoneSwitcherNetworkRequstListener: got request NetworkRequest [ BACKGROUND_REQUEST id=2, [ Transports: CELLULAR Capabili
D PhoneSwitcherNetworkRequstListener: got request NetworkRequest [ REQUEST id=7, [ Capabilities: INTERNET&NOT_RESTRICTED&TR
D PhoneSwitcherNetworkRequstListener: got request NetworkRequest [ REQUEST id=1, [ Capabilities: INTERNET&NOT_RESTRICTED&TR
D CarLatinIME: onCreate
D PhoneSwitcherNetworkRequstListener: got request NetworkRequest [ TRACK_DEFAULT id=18, [ Capabilities: INTERNET&NOT_RESTRI
D PhoneGlobals: Radio technology switched. Now CDMA is active.
I ActivityManager: Finished processing BOOT_COMPLETED for u0
D PhoneGlobals: Radio technology switched. Now GSM is active.
D MainActivity: onCreate
D MediaFragment: onCreateView
D HereNavigationNotificationService: onCreate()
D CarLatinIME: onCreate
D WifiService: Handle boot completed
D Cluster.Service: onCreate
I bootstat: Service started: /system/bin/bootstat --record_boot_complete --record_boot_reason --record_time_since_factory_r
D MainActivity: onCreate
D MediaFragment: onCreateView
D MediaFragment: onCreateView
I AlarmClock: AlarmInitReceiver android.intent.action.LOCKED_BOOT_COMPLETED
D CarLatinIME: onCreate
D VoiceInteractionManagerService: switchImplementation(10) took to complete: 0ms
I ActivityManager: Posting BOOT_COMPLETED user #10
D LucidSystemService: onReceive: intent android.intent.action.USER_UNLOCKED
D LucidSystemService: START LucidCarLauncher
D ONSAutoboot: Received android.intent.action.LOCKED_BOOT_COMPLETED
D LucidSystemService: onStartCommand    :isFteRequiredfalse
D CM.MessengerService: onCreate
D LucidLauncherActivity: onCreate
D LucidSysUiService: onCreate Called
```

1.3.3 IPC AIDL ProfileDataService

- 系统层面整个系统的启动顺序：IPC AIDL ProfileData 进程间服务最先 run 起来!!!

```
I SettingsState: ^^Iat com.android.providers.settings.SettingsProvider.onCreate(SettingsProvider.java:330)
<<<<<<<<<<============ 这里我昨时加入前面一篇的 Settings provider 的部分,
<<<<<<<<<<============ 这里不是搞不懂吗, 它在一个相对更高的层面上注册了 Settings.Global ContentProvider 看这里就够了, 其它可以跳
V SettingsProvider: Notifying for 0: content://settings/global/location_global_kill_switch
V SettingsProvider: Notifying for 0: content://settings/global/location_global_kill_switch

D WifiActiveModeWarden: Switching from WifiDisabledState to WifiDisabled
D VoiceInteractionManagerService: switchImplementation(0) took to complete: 0ms

I CarServiceHelper: Switching to user 10 on boot
I CAR.SERVICE: Service onCreate
I CAR.POWER: User switch disallowed while booting
D CAR.POWER: getTargetUserId(): current=10, target=UserInfo[id=10, name=Primary Driver, flags=ADMIN|INITIALIZED], isTargetP
D CAR.POWER: Not switching to 10 because it's not allowed
I AlarmClock: AlarmInitReceiver android.intent.action.LOCKED_BOOT_COMPLETED

D CarUserService: onCreate()
D VoiceInteractionManagerService: switchImplementation(0) took to complete: 4ms
I ActivityManager: Posting BOOT_COMPLETED user #0 // Headless System user 0
```

```
D CarrierProvider: onCreate 系统级别 Content Provider <<<<<<<<<<<<<<<<<=======================
D CarrierIdProvider: onCreate
W ContextImpl: Calling a method in the system process without a qualified user: <<<<<<<<<<<<<<<<<=======================
android.app.ContextImpl.startService:1570 android.content.ContextWrapper.startService:669
com.lucid.car.profiledata.ProfileDataApp.onCreate:20 android.app.Instrumentation.callApplicationOnCreate:1189
android.app.ActivityThread.handleBindApplication:6460
D ProfileDataService: onStartCommand(): starting service with flags= 0, startId= 1 <<<<<<<<<<<<<<<<<=======================
D ProfileDataService: onStartCommand(): initializing repository...
D ProfileDataService: onStartCommand(): starting random number generator for IM ALIVE messages...
D ProfileDataService: onStartCommand(): service has been started

D ONSAutoboot: Received android.intent.action.LOCKED_BOOT_COMPLETED
D LucidDmsService: onReceive(): android.intent.action.BOOT_COMPLETED
I SystemServiceManager: Calling switchUser u10
```

- 在任何其它应用层的应用启动起来之前，profileDataService 已经先运行起来

1.3.4 系统起来之后前后台的几个应用

```
px3lucid:/ # ps -A | grep -i com.lucid
system        2359   839 5074412  70388 SyS_epoll_wait      0 S com.lucid.car.profiledata // 这个东西看来没看懂，它是 AIDL 相
system        2488   839 5137312 104584 SyS_epoll_wait      0 S com.lucid.userprofile     // 这里定义的是 AIDL 的接口吗？
system        2699   839 5112756  55276 SyS_epoll_wait      0 S com.lucid.car.service
<<<<<<<<<<<<================================ 上面这两个系统层面的 Process 不曾变化，它们为多个用户所共有!!!!!!
u10_system   22962   839 5453316 130716 SyS_epoll_wait      0 S com.lucid.home
u10_system   23364   839 5140988 104888 SyS_epoll_wait      0 S com.lucid.car.carlauncher
u10_system   23389   839 5232280  71396 SyS_epoll_wait      0 S com.lucid.car.sysui
u10_system   23474   839 6006708 146880 SyS_epoll_wait      0 S com.lucid.car.carcontrols
u10_system   23494   839 5178108 112792 SyS_epoll_wait      0 S com.lucid.car.hvac
u10_system   23546   839 5254832 114448 SyS_epoll_wait      0 S com.lucid.car.carsettings
u10_system   23604   839 5095440  69272 SyS_epoll_wait      0 S com.lucid.car.media
u10_system   24319   839 5966052 126072 SyS_epoll_wait      0 S com.lucid.assistant
u10_system   24348   839 4995212  49100 SyS_epoll_wait      0 S com.lucid.car.dialer
u10_system   24420   839 5110648  57140 SyS_epoll_wait      0 S com.lucid.car.service
u10_a46      24605   839 4978492  46748 SyS_epoll_wait      0 S com.lucid.ota_updater
u10_system   24672   839 5005164  48248 SyS_epoll_wait      0 S com.lucid.profilesetup
u10_system   24913   839 4984496  47360 SyS_epoll_wait      0 S com.lucid.userprofile // 这里定义的是 AIDL 的接口的应用层实现
```

- 试验一下，再重新切回到 primary driver 用户，比较一下程序的 pid，为什么这个客户账户没有 com.lucid.userprofile 运行程序??? 可能有的时候切得太早了，或者没有等到它起来就终止了抓取 log

```
px3lucid:/ # ps -A | grep -i com.lucid
system        2359   839 5074412  70644 SyS_epoll_wait      0 S com.lucid.car.profiledata
system        2488   839 5137312 104088 SyS_epoll_wait      0 S com.lucid.userprofile
system        2699   839 5112756  55276 SyS_epoll_wait      0 S com.lucid.car.service
<<<<<<<<<<<<================================ 上面这两个系统层面的 Process 不曾变化，它们为多个用户所共有!!!!!!
u11_system   27325   839 5441284 111640 SyS_epoll_wait      0 S com.lucid.home
u11_system   27757   839 5298088  69048 SyS_epoll_wait      0 S com.lucid.car.sysui
u11_system   27768   839 5189612 105200 SyS_epoll_wait      0 S com.lucid.car.carlauncher
u11_system   27883   839 6075716 147376 SyS_epoll_wait      0 S com.lucid.car.carcontrols
u11_system   27908   839 5244028 112376 SyS_epoll_wait      0 S com.lucid.car.hvac
u11_system   27932   839 5305712 113716 SyS_epoll_wait      0 S com.lucid.car.carsettings
u11_system   28008   839 5112684  59640 SyS_epoll_wait      0 S com.lucid.car.media
u11_system   28660   839 5229928  61940 0                   0 R com.lucid.assistant
u11_system   28689   839 5061148  50100 SyS_epoll_wait      0 S com.lucid.car.dialer
u11_system   28772   839 5176072  56832 SyS_epoll_wait      0 S com.lucid.car.service
u11_a46      28961   839 5044428  46804 SyS_epoll_wait      0 S com.lucid.ota_updater
```

- 再切一遍

```
  px3lucid:/ # ps -A | grep -i lucid
system        2359   839 5074412  71400 SyS_epoll_wait      0 S com.lucid.car.profiledata
system        2488   839 5137312 104648 SyS_epoll_wait      0 S com.lucid.userprofile
system        2699   839 5130196  55480 SyS_epoll_wait      0 S com.lucid.car.service
<<<<<<<<<<<<================================ 上面这两个系统层面的 Process 不曾变化，它们为多个用户所共有!!!!!!
u10_system   30343   839 5456292 111996 SyS_epoll_wait      0 S com.lucid.home
u10_system   30789   839 5141116 105260 SyS_epoll_wait      0 S com.lucid.car.carlauncher
u10_system   30823   839 5232152  69196 SyS_epoll_wait      0 S com.lucid.car.sysui
u10_system   30932   839 6006852 147084 SyS_epoll_wait      0 S com.lucid.car.carcontrols
```

```
u10_system   30961   839 5178236 112952 SyS_epoll_wait      0 S com.lucid.car.hvac
u10_system   31005   839 5257392 114532 SyS_epoll_wait      0 S com.lucid.car.carsettings
u10_system   31081   839 5081676  58544 SyS_epoll_wait      0 S com.lucid.car.media
u10_system   31703   839 5944888 113684 SyS_epoll_wait      0 S com.lucid.assistant
u10_system   31734   839 4995212  49180 SyS_epoll_wait      0 S com.lucid.car.dialer
u10_system   31788   839 5128088  56140 SyS_epoll_wait      0 S com.lucid.car.service
u10_a46      31995   839 4978492  46740 SyS_epoll_wait      0 S com.lucid.ota_updater
u10_system   32042   839 5005164  48388 SyS_epoll_wait      0 S com.lucid.profilesetup
u10_system   32328   839 5067744  47716 SyS_epoll_wait      0 S com.lucid.userprofile // 这里定义的是 AIDL 的接口的应用层实现
```

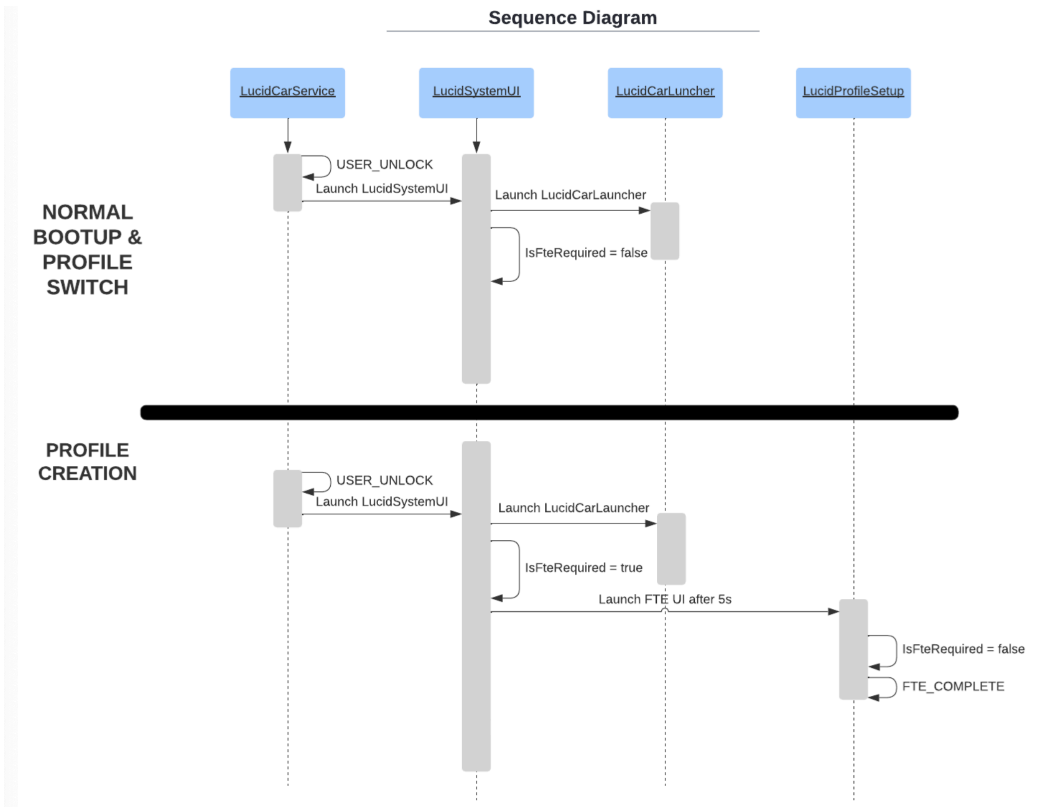
- 再切一次

```
px3lucid:/ # ps -A | grep -i lucid
system        2359   839 5074412  71816 SyS_epoll_wait      0 S com.lucid.car.profiledata
system        2488   839 5137312 104888 SyS_epoll_wait      0 S com.lucid.userprofile
system        2699   839 5147892  55772 SyS_epoll_wait      0 S com.lucid.car.service
<<<<<<<<<<================================== 上面这两个系统层面的 Process 不曾变化，它们为多个用户所共有!!!!!!
u11_system    1090   839 5424052 112696 SyS_epoll_wait      0 S com.lucid.home
u11_system    1801   839 5140988 104932 SyS_epoll_wait      0 S com.lucid.car.carlauncher
u11_system    1833   839 5232152  69016 SyS_epoll_wait      0 S com.lucid.car.sysui
u11_system    1968   839 6006852 146980 SyS_epoll_wait      0 S com.lucid.car.carcontrols
u11_system    2004   839 5195676 112716 SyS_epoll_wait      0 S com.lucid.car.hvac
u11_system    2032   839 5274832 114408 SyS_epoll_wait      0 S com.lucid.car.carsettings
u11_system    2138   839 5081676  58716 SyS_epoll_wait      0 S com.lucid.car.media
u11_system    3085   839 5953048 113116 SyS_epoll_wait      0 S com.lucid.assistant
u11_system    3116   839 4995212  49204 SyS_epoll_wait      0 S com.lucid.car.dialer
u11_system    3198   839 5110648  56152 SyS_epoll_wait      0 S com.lucid.car.service
u11_a46       3471   839 4961052  46876 SyS_epoll_wait      0 S com.lucid.ota_updater
u11_system    3522   839 5005164  48380 SyS_epoll_wait      0 S com.lucid.profilesetup
u11_system    3803   839 5001936  47340 SyS_epoll_wait      0 S com.lucid.userprofile // 这里终于出来了，可是什么情况下会有，
```

- 还是说我只能 ps 出在前台运行的服务？这方面的命令要再熟悉一下。

1.4 Use-cases

- Normal bootup sequence
- Profile switch
- Profile creation



- Options to start LucidSystemUI
  - Start LucidSystemUI from LucidCarService app by listening to USER_UNLOCKED
  - Start LucidSystemUI from LucidUserProfile app by listening to USER_UNLOCKED
  - Make LucidSystemUI act as Android SystemUI
  - Our long-term goal is to achieve option-3, as it is a more ideal way to start the LucidSystemUI app.

1.5 Limitations of current architecture

- LucidSystemUI & LucidCarLauncher apps both are relying on BOOT_COMPLETED events, which can be delayed due to some reasons, and in that scenario, it can reproduce some issues like a user is able to see a black screen or user is not able to interact with the screen.

- There is no centralized place to control the sequence of these apps with a particular order of which app should launch first.

1.6 Pros and cons of a new architecture design

1.6.1 Pros

- With the new architecture, we will get LucidSystemUI as a new centralized place to control all sequences and we can control the order of various apps by one app.

- Provide ease to debug any issues in the future

- As BOOT_COMPLETED won't be used to invoke UI on CID or ICR, the timings will remain the same.

1.6.2 Cons

- If LucidSystemUI crashes, there won' t be any UI seen on CID.

2 LucidCarService

- 最先启动（次于后于 ProfileData IPC AIDL 进程间服务），但任何用户下都有一个 instance

2.1 AndroidManifest.xml

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
          coreApp="true"
          package="com.lucid.car.service"
          android:sharedUserId="android.uid.system"> <!-- 开发了多个 APK 并且需要他们之间互相共享资源 -->

  <protected-broadcast android:name="com.lucid.carlib.ACTION_POWER_STATE" /> <!-- 这个是定义在模块里面？ -->
  <!-- 安卓系统全局广播 -->
  <uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED" />
  <uses-permission android:name="android.permission.INTERACT_ACROSS_USERS" />
  <!-- 下面这两个：广播由本应用发出，由 LucidUserProfile 应用的 LucidDmsService 在接听 -->
  <protected-broadcast android:name="com.lucid.userprofile.ACTION_LOGIN_TO_USER_PROFILE"/>
  <protected-broadcast android:name="com.lucid.userprofile.ACTION_LOGOUT_FROM_USER_PROFILE"/>
  <application
      android:name=".LucidCarServiceApp"
      android:label="@string/app_name"
      android:directBootAware="true"
      android:persistent="true">
    <activity
        android:name=".fallbackhome.MainActivity" <!-- 占位符似的空类，定义为空 -->
        android:launchMode="singleInstance"
        android:configChanges="touchscreen"
        android:theme="@android:style/Theme.Translucent.NoTitleBar">
      <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.HOME" />
      </intent-filter>
    </activity>
    <activity
        android:name=".fallbackhome.CidMainActivity" <!-- 占位符似的空类，定义为空 -->
        android:launchMode="singleInstance"
        android:configChanges="touchscreen"
        android:theme="@android:style/Theme.Translucent.NoTitleBar">
      <intent-filter>
        <action android:name="android.intent.action.MAIN" />
      </intent-filter>
    </activity>
    <service android:name=".LucidSystemService" />
    <service
        android:name=".LucidProfileSessionService"
        android:enabled="true"
        android:exported="false" />
    <service
        android:name=".LucidTimeZoneService"
        android:enabled="true"
        android:exported="false" />
    <receiver
        android:name=".LucidCarServiceApp$OnBootReceiver"
        android:enabled="true"
        android:exported="true">
      <intent-filter>
        <action android:name="android.intent.action.BOOT_COMPLETED" />
      </intent-filter>
    </receiver>
    <receiver
        android:name=".ota.InjectBaseVersionS2110"
        android:enabled="true"
        android:exported="true">
      <intent-filter>
        <action android:name="android.intent.action.BOOT_COMPLETED" />
      </intent-filter>
    </receiver>
  </application>
</manifest>
```

2.2 LucidCarServiceApp.java

- 这个类的静态成员 LucidCarServiceApp.OnBootReceiver 接收器是专用来接收 BOOT_COMPLETED 广播的，定义在上面

```
<receiver android:name=".LucidCarServiceApp$OnBootReceiver"
    android:enabled="true"
    android:exported="true">
  <intent-filter> <action android:name="android.intent.action.BOOT_COMPLETED" />
  </intent-filter>
</receiver>
```

- 因为这个类是系统 BOOT_COMPLETED 之后的入口程序之一，而我们的逻辑定义为先从系统用户 boot 完后再切换到用户空间
- 记得前面看见过，LucidUserProfile 应用同样监听这个广播 BOOT_COMPLETED，但是因为它的接收优先级最高（3000），所以应该是那个应用最先启动！
- 同样监听系统 BOOT_COMPLETED 的还有 ota.InjectBaseVersionS2110 接收器，这里不表
- 这里有段它们写的关于除系统用户之外，如果 primary, 2ndary, & Guest 用户出现的时候，系统的大致情况

```
 * One instance of LucidCarServiceApp is spun up under system user.
 * The instance's life cycle is tied to system, is used to launch car power management, and etc....
 *   This is done by setting
 *   1. AndroidManifest's coreApp="true"
 *   2. android:sharedUserId="android.uid.system".
 *
 * In addition, one more instance is spun up for each 2ndary user profile.
 * This instance's life cycle is tied to user profile login session,
 * and is triggered by listening to BOOT_COMPLETE broadcast intent.
 *
 * Launching of various services for both of cases is done inside application object's onCreate() initialization.
```

- LucidCarServiceApp.java：这里面还有很多的逻辑没有看懂

```
public class LucidCarServiceApp extends Application { // LUCID 系统 BOOT_COMPLETED 后的启动程序入口 ???????
    public static final String TAG = "LucidCarServiceApp";
    private SetClockController mSetClockController;
    @Override
        public void onCreate() {
        super.onCreate();
        Log.d(TAG, "onCreate...");
// 这里分用户往下执行：系统用户，或真正的用户
        if ( Process.myUserHandle().equals( UserHandle.SYSTEM ) )
            startLucidSystemServices(); // 执行 Boot 相关的通用/或系统维护程序
        else
            startLucidProfileSessionServices(); // 切换到用户空间/执行相关
    }
    void startLucidSystemServices() {
        Log.d( TAG, "startLucidSystemServices()" );
// Listen to TCU epoch signal to set CCC clock.
        mSetClockController = new SetClockController();
        mSetClockController.init( this, new SetClockController.ClockListener() {
                @Override
                    public void onClockUpdated() {
                    mSetClockController.cleanup();
                    mSetClockController = null;
                }
            } );
        Log.d(TAG, "starting PowerManagementService...");
        startService( new Intent( this, LucidSystemService.class ) ); //  《《==== LucidSystemService

        ContentResolver contentResolver = getContentResolver();
        boolean isTimeZoneAuto = Feature.isEnabled(contentResolver, Settings.Global.AUTO_TIME_ZONE, Feature.ENABLED);
        if (isTimeZoneAuto) {
            Log.d(TAG, "starting LucidTimeZoneService...");
            startService( new Intent( getApplicationContext(), LucidTimeZoneService.class ) );
        }
        contentResolver.registerContentObserver(
            Settings.Global.getUriFor( Settings.Global.AUTO_TIME_ZONE ), false, autoTimeZoneObserver );
    }
    private ContentObserver autoTimeZoneObserver = new ContentObserver(new Handler()) {
        @Override
        public void onChange(boolean selfChange) {
            super.onChange(selfChange);
            // ...
        }
    };
    void startLucidProfileSessionServices() {
        Log.d( TAG, "startLucidProfileServices()" );
// Kick off ProfileSessionService
        startService( new Intent( this, LucidProfileSessionService.class ) );
    }

//================================================================
// BOOT_COMPLETE Handling.
//================================================================
/**
 * An (almost) empty broadcast receiver to catch BOOT_COMPLETE to trigger
 * spinning up of application object for 2ndary user profile sessions.
 *
 * Also used to trigger exec of 'udscat' in getEcuVersions(): 这里还看不懂
 */
    public static class OnBootReceiver extends BroadcastReceiver {
        @Override public void onReceive(Context context, Intent intent) {
            Log.d( TAG, "OnBootReceiver.onReceive()" );
            getEcuVersions();
        }
    }
 // * In Developer Builds, execs udscat binary to fetch ECU version information and send to IVIMICRO services.
    private static void getEcuVersions() {
        if (Feature.DEVELOPER_BUILD && Process.myUserHandle().equals(UserHandle.SYSTEM)) {
            Log.d(TAG, "getEcuVersions - exec udscat");
            try {

                Runtime.getRuntime().exec("udscat"); // 那么执行这个命令的作用是什么呢？Collects ECU identifiers
// Collects ECU identifiers based on ECU configuration provided in ecus.json. Outputs the list in JSON.
// https://gitlab.corp.atieva.com/sftdts/udscat
            } catch (Exception e) {
                Log.d(TAG, "getEcuVersions - " + e.toString());
            }
            Log.d(TAG, "getEcuVersions - exec udscat done");
        }
    }
}
```

2.3 LucidSystemService: 关系 LucidCarLauncher, LucidSystemUI 以及可能的 LucidProfileSetup（用户不曾注册）应用的启动

- 通过监听几个系统级别用户 boot 事件(来自安卓 10 多用户的系统事件?ACTION_USER_UNLOCKED 和 ACTION_USER_STOPPED)，来启动用户空间的几个应用程序

```
// public class LucidSystemService extends Service {
private static final String TAG = "LucidSystemService";
private static final int PROFILE_SETUP_APP_DELAY = 5000; // 5 秒
// Intent for user_starting and user_stopping: 这些事件的广播者应该是安卓系统
private static final String ACTION_USER_UNLOCKED = "android.intent.action.USER_UNLOCKED";
private static final String ACTION_USER_STOPPED = "android.intent.action.USER_STOPPED";
private static final String EXTRA_USER_HANDLE ="android.intent.extra.user_handle";
```

```
// Intent to start LucidSysUiService
private static final String IS_FTE_REQUIRED = "IS_FTE_REQUIRED";

//================================================================ 系统应用层 几个程序启动
private void registerReceiver() {
    IntentFilter filter = new IntentFilter();
    filter.addAction( ACTION_USER_UNLOCKED );
    filter.addAction( ACTION_USER_STOPPED );
    getApplicationContext().registerReceiverAsUser( mReceiver, UserHandle.ALL, filter, (String)null, (Handler)null );
}
// 这里再次出现: 即使是 getApplicationContext(), 同样需要 unregisterReceiver () 以防泄露
private void unregisterReceiver() {
    unregisterReceiver( mReceiver );
}
// 通过监听几个系统级别用户 boot 事件, 来启动用户空间的几个应用程序
private BroadcastReceiver mReceiver = new BroadcastReceiver() {
    @Override
    public void onReceive(Context context, Intent intent) {
        if (ACTION_USER_UNLOCKED.equals(intent.getAction())) { // <<<<====== ACTION_USER_UNLOCKED
            if (intent.hasExtra(EXTRA_USER_HANDLE)){
                int userId = intent.getIntExtra(EXTRA_USER_HANDLE, -1);
// start LucidCarLauncher
                Intent launchIntent = new Intent(Intent.ACTION_MAIN);
                launchIntent.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
                launchIntent.setComponent(new ComponentName(LucidSystemUiUtil.LUCID_CAR_LAUNCHER_PACKAGE,
                                                            LucidSystemUiUtil.LUCID_CAR_LAUNCHER_COMPONENT));
                Log.d(TAG, "START LucidCarLauncher");
                LucidDisplays.startActivityForGivenUser(context, LucidDisplays.DISPLAY_CID, launchIntent, UserHandle.of(use
// start LucidSysUiService
                context.startForegroundServiceAsUser(
                    new Intent()
                    .setClassName(LucidSystemUiUtil.LUCID_SYS_UI_SERVICE, LucidSystemUiUtil.LUCID_SYSTEM_UI_COMPONENT)
                    .setAction(LucidSystemUiUtil.ACTION_LAUNCH_CID), UserHandle.of(userId));
// Launch FTE if required
                boolean launchProfileSetup = isFteRequired(context);
                Log.d(TAG, "onStartCommand    :isFteRequired" + launchProfileSetup);
                if (launchProfileSetup)
                    launchLucidProfileSetupApp(userId);
            }
        }
    }
};
// Launching FTE app after USER_UNLOCKED event with 5s delay
private void launchLucidProfileSetupApp(int userProfileId) { // LucidProfileSetup 应用
    Log.d(TAG, "launchLucidProfileSetupApp");
    new Handler().postDelayed(new Runnable() {
            @Override public void run() {
                Intent launchIntent = new Intent(Intent.ACTION_MAIN);
                launchIntent.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
                launchIntent.setComponent(new ComponentName(LucidSystemUiUtil.LUCID_PROFILE_SETUP_PACKAGE,
                                                            LucidSystemUiUtil.LUCID_PROFILE_SETUP_COMPONENT));
                Log.d(TAG, "START profile setup");
                startActivityAsUser(launchIntent, UserHandle.of(userProfileId));
            }
        }, PROFILE_SETUP_APP_DELAY);
}
private boolean isFteRequired(Context context) {
    String value = Settings.Global.getString( context.getContentResolver(), IS_FTE_REQUIRED );
    return Boolean.parseBoolean(value);
}
//================================================================
// Life cycle
//================================================================
private PowerStateMachine mPowerStateMachine;
private OtaManager mOtaManager;
private VinPresenter vinPresenter;
private IccAlertsController iccAlertsController;
@Override
public void onCreate() {
    super.onCreate();
    registerReceiver();
    mPowerStateMachine = new PowerStateMachine();
    mPowerStateMachine.init( this );
    vinPresenter = new VinPresenter(this);
    vinPresenter.initControllerAndListen();
```

```java
// Bluetooth.
    initBluetoothName();
    mOtaManager = new OtaManager( this );
    iccAlertsController = new IccAlertsController();
    iccAlertsController.init(this, mIccAlertsListener );
    DayNightModeController dayNightModeController = new DayNightModeController();
    dayNightModeController.persistAutoBrightness(this, true);
}
@Override
public void onDestroy() {
    super.onDestroy();
    unregisterReceiver();
    iccAlertsController.cleanup();
    iccAlertsController = null;
    mOtaManager.cleanup();
    mOtaManager = null;
    vinPresenter.cleanup();
    vinPresenter = null;
    mPowerStateMachine.cleanup();
    mPowerStateMachine = null;
}
//================================================================
// Bluetooth init...
//================================================================
private void initBluetoothName() {
    LocalBluetoothManager localBtManager = LocalBluetoothManager.getInstance(getApplicationContext(), /* onInitCallback= */
    if ( localBtManager != null && localBtManager.getBluetoothAdapter() != null ) {
        String btName = "Lucid Air " + VehicleInfoSettings.getVINNumber( getContentResolver() );
        localBtManager.getBluetoothAdapter().setName( btName );
    } else Log.d(TAG, "Bluetooth is not supported on this device");
}
//================================================================
// ICC Alerts
//================================================================
public static final String SETTINGS_GLOBAL_ALERT_LIST = "SETTINGS_GLOBAL_ALERT_LIST";
private final SparseIntArray messages = new SparseIntArray();
private IccAlertsController.IccAlertsListener mIccAlertsListener = // 基类定义为 abstract, 所以这里不用 @Override
    new IccAlertsController.IccAlertsListener() {
    public void onIccAlertMessageW(int alertId) {
        StringBuilder str = new StringBuilder();
        if (alertId != 0 && messages.get(alertId, -1) < 0) {
            messages.put(alertId, (int) System.currentTimeMillis());
        }
        for (int i = 0; i < messages.size(); i++) {
            str.append(messages.keyAt(i));
            str.append(",");
            str.append(messages.valueAt(i));
            if (i + 1 < messages.size())
                str.append(",");
        }
        Settings.Global.putString(getContentResolver(), SETTINGS_GLOBAL_ALERT_LIST, str.toString());
    }
};
```

2.4 LucidProfileSessionService.java：由车硬件向 Lucid Air OS 软件系统配置，面向所有非系统用户; 现在，可以看见更高的天空了吗？!!!

- 关系用户登录登出，lucid air 安卓系统的机理是怎样的呢?

```
当 cold boot, 系统用户登录是我们要求的
系统用户登录完成后, 自家扫描到的用户登录也是我们定义要求的
与此同时, 硬件方面, 我们在发什么样的信号, 这些信号如何帮助我们实现期望逻辑的推进:
比如, 当自家用户登录后, 我们是否启动 lane departure power 等等
这些个同户登录后的硬件启动逻辑是在这个服务里实现
```

- 这里涉及到 ProfileData 跨进程模块，有些接口和服务都是在进程间通信: 这里还需要弄明白
- 还记得 lucid-profile-lib 定义了一个服务建立联接之后的回调的公用接口吗? 感觉是一个简单的进程间服务建立联接后的回调，用来设置一个标志位: mIsProfileDataReady, 从而来根据这个存在本地的标志位来判断是否加载当前用户相关的某些初始化配置

```
// public class ProfileData { // lucid-profile-lib
// 谁在实现这个接口: LucidCarService 应用中的 LucidProfileSessionService.java 实现和实例化了这个接口
// 谁在实现这个接口? 所有想要使用 ProfileData instance 实例的类与服务, 都需要实现接个接口类
public interface ServiceConnectionListener {
    void onConnected();
    void onDisconnected();
}
```

- 这里，就有成员变量实例化了这个接口，进而实现服务建立与否的监听回调:

```
profileData = new ProfileData( getContext() );
```

- 上面这一个实例化，是如何实现与远程进程间服务自动建立联接的?
- 同时，又是如何与 LucidProfileData 应用建立同步联接的? 感觉这两步的逻辑还不懂。。。。。。

```
This version of ProfileData (as of 27 APR 2021) is a step toward objectifying access to
persist/retrieve methods, which up till now have been implemented only as static methods.

To use ProfileData it *must first be instantiated*, e.g.:
profileData = new ProfileData( getContext() );
The binding to ProfileDataService takes place upon instantiation, within the constructor.
```

- 现在，再来读这个服务，可以看见更高的天空了吗? 可是，为什么当年的我，几个周之前的我，读这些代码的时候，就是腾云驾雾/云里雾里的状态呢?

```
public class LucidProfileSessionService extends Service {
    private static final String TAG = "ProfileSessionService";
    private static final int PROFILE_READY_TIME = 2 * 1000;

    private static final String KEY_MARKET_VALUE = "KEY_MARKET_VALUE";
    private static final String KEY_REGION_VALUE = "KEY_REGION_VALUE";
//==================================================================
// Life cycle
//==================================================================
    private ProfileData mProfileData;
    private boolean mIsProfileDataReady;
    private boolean isDataDirty;
    // private Handler mHandler;

    @Override public void onCreate() {
        super.onCreate();
        mHandler = new Handler();
// IProfileDataService.java, LucidProfileDataApp 同时绑定服务与实例化应用吗？这个过程很神奇呀。。。。。。
        mProfileData = new ProfileData(this);
        mProfileData.bindService(mProfileDataListener); // 定义在上面: 这里还是想得不够清楚,

        // mAdasController = new AdasController();
        // 没有搞明白接下来的内容为什么与用户数据发生了交叉
        // this is a hack until Joseph's check in is available to grab this from profile lib
        int ldpLevel = Settings.Global.getInt(getApplicationContext().getContentResolver(),
                                              ProfileKey.SETTINGS_ADAS_LDP_LEVEL, AdasController.LDP_LEVEL_INTERVENTION);
        mAdasController.init(this, new AdasControllerListener(), ldpLevel);
        initializeSystemDefaultUnit();
    }
    @Override public void onDestroy() {
        mProfileData.unbindService();
    }
    @Override public IBinder onBind(Intent intent) {
        return null;
    }
// 绑定跨进程服务的服务联接监听回调, 重中之重 !!!
    private final ProfileData.ServiceConnectionListener mProfileDataListener =
        new ProfileData.ServiceConnectionListener() {
        @Override
        public void onConnected() {
            mIsProfileDataReady = true;
            persistHAWProfile();
        }
        @Override
        public void onDisconnected() {
            mIsProfileDataReady = false;
        }
    };
    private void persistHAWProfile() {
        if (mIsAdasControllerReady && mIsProfileDataReady && isDataDirty) {
            /**
             * We will save Integer instead of boolean into profileData:
             * 1, For guest user, logservice will send 1 to HWA_USR_PROF_SETTING
             * 2, For new user, logservice will send 1 to HWA_USR_PROF_SETTING no matter if HWA is active or inactive
             * 3, For existing user, logservice will send 1 / 0 to HWA_USR_PROF_SETTING based on HWA status.
             * */
// 为什么需要把这个东西写进数据库，有什么关联？HWA 是什么意思呢？LucidCarSettings，可能涉及各用户自定义硬件配置吧
            mProfileData.persistInt(ProfileKey.SETTINGS_ADAS_HWA_STATUS, mAdasController.isHWAEnabled() ? HWA_PROFILE_ENABL
            isDataDirty = false;
        }
    }
    /**
     * System default unit will be decided by system configuration(IMarket & IRegion)
     * CCC is currently supporting units as below, will add more in the future if needed.
     * 1, DistanceUnit | 2, Temperature | 3, Time Format |4, Tire Pressure
     */
    private void initializeSystemDefaultUnit() {
        String setting = getString(CarSettings.IS_DISTANCE_MI_UNIT);
        String isImperialUnit = getString(UnitPrefController.IS_DISTANCE_IMPERIAL_MI_UNIT);
        if (TextUtils.isEmpty(setting) || TextUtils.isEmpty(isImperialUnit)) {
            UnitPrefController unitPrefController = new UnitPrefController();
// 这里实现的是：由车 CAR 硬件向软件的同步，即保持软件安卓个性化 OS 与车的硬件配置同步
            unitPrefController.init(this, new UnitPrefController.UnitPrefListener() {
                    @Override public void onVhalConnectedW() {
                        if (unitPrefController.hasValidMarket()
                            || unitPrefController.hasValidRegion()) {
                            persistSystemUnit(unitPrefController);
                        }
                    }
                    @Override public void onDistanceUnitReady(boolean isDistanceInMiles) {
                        persistSystemUnit(unitPrefController);
                        // region & market are vehicle based properties, don't need to save by using user based key
                        UnitPrefController.Market defaultMarket = unitPrefController.getDefaultMarket();
                        if (defaultMarket != UnitPrefController.Market.UNKNOWN)
                            Settings.Global.putString(getContentResolver(), KEY_MARKET_VALUE, defaultMarket.value);
                        UnitPrefController.Region defaultRegion = unitPrefController.getDefaultRegion();
                        if (defaultRegion != UnitPrefController.Region.UNKNOWN) {
                            Settings.Global.putString(getContentResolver(), KEY_REGION_VALUE, defaultRegion.value);
                        }
                        Log.d(TAG, "onDistanceUnitReady is called, market:" + defaultMarket + ", region:" + defaultRegion);
                    }
                });
        }
    }
// LucidCarSettings: Displays Fragement ==> UnitsFragment ==> BatteryDisplay + Distance + TimeFormat + Temperature 值的提取
    private void persistSystemUnit(UnitPrefController unitPrefController) {
        if (unitPrefController == null)  return;
        boolean isDistanceInMiles = unitPrefController.isDistanceUnitInMiles();
        Log.d(TAG, "Persist distance unit, value: " + isDistanceInMiles);
        putString(CarSettings.IS_DISTANCE_MI_UNIT, String.valueOf(isDistanceInMiles));
// 调用所有应用层回调，要求应用层所有监听者与硬件同步
        unitPrefController.updateDistanceDisplayUnit(isDistanceInMiles ? 0 : 1);
        if (unitPrefController.hasValidMarket()) {
            UnitPrefController.Market defaultMarket = unitPrefController.getDefaultMarket();
            boolean isUKMarket = unitPrefController.isUKMarket(); // 特例化 UK，我还需要特例化 EU 不是吗？你找到源头了吗，我的
            Log.d(TAG, "defaultMarket: " + defaultMarket);
// 由车配置硬件，向个性化安卓 OS 系统软件同步：写入 Lucid Air OS 全局 Settings.Global ContentProvider
            persistUnitDefaultValueInGlobalSettings(UnitPrefController.IS_DISTANCE_IMPERIAL_MI_UNIT, isUKMarket);
            persistUnitDefaultValueInGlobalSettings(CarSettings.IS_TIME_24_HR, defaultMarket.timeFormat.value);
            persistUnitDefaultValueInGlobalSettings(CarSettings.IS_TEMP_F_UNIT, defaultMarket.temperatureFormat.value);
            persistUnitDefaultValueInGlobalSettings(CarSettings.IS_PRESSURE_PSI_UNIT, defaultMarket.tirePressureFormat.valu
        }
// 同车配置硬件，向用户个性化配置写入数据，因为在 LucidCarService 应用中配置这些，所以 Feature 说需要经历一次 PowerCycle，现在理解
        if (mIsProfileDataReady)
            persistUnitDefaultValueInProfileData(unitPrefController);
        else {
            mHandler.postDelayed(() -> {
                    if (mIsProfileDataReady)
                        persistUnitDefaultValueInProfileData(unitPrefController);
                    else
                        Log.d(TAG, "Persist unit default values error, profile is not available");
```

```java
                }, PROFILE_READY_TIME); // 延迟 2 秒
            }
        }

        private void persistUnitDefaultValueInGlobalSettings(String name, boolean value) {
            persistGlobalSettings(name, String.valueOf(value));
        }
        private void persistGlobalSettings(String name, String value) {
            putString(name, value);
        }
    // 我在这里需要个性化配置的，也就包括了面向 EU 向户的，ADB 的启用，需要添加这一条
        private void persistUnitDefaultValueInProfileData(UnitPrefController unitPrefController) {
            boolean isDistanceInMiles = unitPrefController.isDistanceUnitInMiles();
            mProfileData.persistBoolean(CarSettings.IS_DISTANCE_MI_UNIT, isDistanceInMiles);
            if (unitPrefController.hasValidMarket()) {
                UnitPrefController.Market defaultMarket = unitPrefController.getDefaultMarket();
                mProfileData.persistBoolean(CarSettings.IS_TIME_24_HR, defaultMarket.timeFormat.value);
                mProfileData.persistBoolean(CarSettings.IS_TEMP_F_UNIT, defaultMarket.temperatureFormat.value);
                mProfileData.persistBoolean(CarSettings.IS_PRESSURE_PSI_UNIT, defaultMarket.tirePressureFormat.value);
            }
        }
        private String getString(@NonNull String key) {
            return CarSettings.getString(this, key);
        }
        private void putString(@NonNull String key, @NonNull String value) {
            CarSettings.putString(this, key, value);
        }
    // lucid-settings-lib 车的配置，写入全局 Settings.Global ContentProvider
    // package com.lucid.carlib.settings.carprefs;
    public static void putString(@NonNull Context context, @NonNull String key, @NonNull String value) {
        Settings.Global.putString(context.getContentResolver(), getUserBasedKey(context, key), value);
    }
    }
```

2.5 PowerStateController

```java
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;
private CarPropertyManager mPropertiesManager;
private final ServiceConnection mCarConnectionCallback = new ServiceConnection() {
    @Override public void onServiceConnected(ComponentName name, IBinder service) {
        try {
            mPropertiesManager = (CarPropertyManager) mCarApiClient
            .getCarManager( android.car.Car.PROPERTY_SERVICE );
            for ( int vPropId : S_LISTEN_TO_VPROP_IDS )
                mPropertiesManager.registerCallback( mPropertiesListener, vPropId, 100 );
        } catch ( android.car.CarNotConnectedException e ) {
            Log.e(TAG, "Car not connected in onServiceConnected");
        }
        // Notify
        if ( mPowerStateListener != null )
            mPowerStateListener.onVHalConnectedW();
    }
    @Override
    public void onServiceDisconnected(ComponentName name) {
        for ( int vPropId : S_LISTEN_TO_VPROP_IDS )
            mPropertiesManager.unregisterCallback( mPropertiesListener, vPropId );
        if ( mPowerStateListener != null )
            mPowerStateListener.onVHalDisconnectedW();
        // Retry connection again in some interval.
    }
};
//===============================================================
// VHAL Properties Callback.
//===============================================================
private CarPropertyManager.CarPropertyEventCallback mPropertiesListener =
    new CarPropertyManager.CarPropertyEventCallback() {
    @Override public void onChangeEvent(CarPropertyValue carPropertyValue) {
        if (carPropertyValue.getPropertyId() == VCU_VEHICLE_POWER_STATE ) {
            if (mPowerStateListener != null) {
                int powerState = (int) carPropertyValue.getValue();
                mPowerStateListener.onUpdatePowerStateW( powerState );
            }
        }
    }
    @Override
    public void onErrorEvent(int i, int i1) {
        Log.e( TAG, "onErrorEvent: i=" + i + ", i1=" + i1 );
    }
};
//===============================================================
// CCC Suspend to RAM
//===============================================================
public static final int CCC_POWER_STATE_UNDEF = 0;  // Not used.
public static final int CCC_POWER_STATE_DISPLAYS_ON = 1;
public static final int CCC_POWER_STATE_DISPLAYS_OFF = 2;
public static final int CCC_POWER_STATE_SUSPEND_TO_RAM = 3;
/**
 * Communicate to VHAL to trigger CCC suspend to RAM sequence with CPMS.
 * @param cccPowerState CCC_POWER_STATE_DISPLAYS_ON    - for ACCESSORY, DRIVE, LIVE_CHARGE, LIVE_UPDATE,
 *                      CCC_POWER_STATE_DISPLAYS_OFF   - for WINK, SLEEP_UPDATE, SLEEP_CHARGE, MONITOR,
 *                      CCC_POWER_STATE_SUSPEND_TO_RAM - for SLEEP, CLOUD1, CLOUD2
 */
public void setCCCPowerState(int cccPowerState) { // PowerStateMachine 会调用这个方法来调度状态
    if (mPropertiesManager == null ) {
        Log.w( TAG, "setCCCPowerState() : Not connected to CarApiClient." );
        return;
    }
    if (cccPowerState != CCC_POWER_STATE_UNDEF
         && cccPowerState != CCC_POWER_STATE_DISPLAYS_ON
         && cccPowerState != CCC_POWER_STATE_DISPLAYS_OFF
         && cccPowerState != CCC_POWER_STATE_SUSPEND_TO_RAM ) {
        Log.w( TAG, "setCCCPowerState() : invalid cccPowerState : " + cccPowerState );
        return;
    }
    Log.d( TAG, "setCCCPowerState() : CCC_POWER_STATE=" + (
                cccPowerState == 0 ? "UNDEF" :
                cccPowerState == 1 ? "DISPLAYS_ON" :
                cccPowerState == 2 ? "DISPLAYS_OFF" :
```

```java
                cccPowerState == 3 ? "SUSPEND_TO_RAM" :
                "UNKNOWN" ) );
        mPropertiesManager.setIntProperty( CCC_POWER_STATE_REQ, 0, cccPowerState );
    }
    //=================================================================
    // Car Property IDs
    //=================================================================
    // Maps to ARXML signal /InfoT_Chassis/PM_VEHPWRST/IVSM_VehPwrSt
    //   Sleep:0, Wink:1, Accessory:2, Drive:3, Live_Charge:4, Sleep_Charge:5,
    //   Live_Update:6, Sleep_Update:7, Cloud1:8, Cloud_2:9, Monitor:10,
    private static final int VCU_VEHICLE_POWER_STATE = 0x21401601;

    // Undef:0, DisplaysOn:1, DisplaysOff:2, SuspendToRAM:3
    private static final int CCC_POWER_STATE_REQ = 0x21401000;

    private final static int[] S_LISTEN_TO_VPROP_IDS = new int [] {
        VCU_VEHICLE_POWER_STATE,
    };
}
```

2.6 PowerStateMachine.java: 这里把它折成几个自己理解的版块来理解

- 定义了抽象类 State，并定义电源的 10 个有效状态

```java
//=================================================================
// State machine
//=================================================================
private State mCurrState = null;
private Handler mHandler = new Handler( new Handler.Callback() {
        @Override public boolean handleMessage(Message message) {
            if (mCurrState != null)
                mCurrState.processMessage(message); // 将它鉴定为电源状态机中的某个有效状态？
            return true;
        }
    } );
abstract class State { // 抽象类：定义三个方法
    void enter() { }
    void exit() { }
    void processMessage(Message message) { // 将它鉴定为电源状态机中的某个有效状态？
        Log.d( TAG, "processMessage(): message=" + message.what );
        // Always force a state transition.
        transitionTo(
            message.what == PowerStateController.POWER_STATE_SLEEP ? mSleepState :
            message.what == PowerStateController.POWER_STATE_WINK ? mWinkState :
            message.what == PowerStateController.POWER_STATE_ACCESSORY ? mAccessoryState :
            message.what == PowerStateController.POWER_STATE_DRIVE ? mDriveState :
            message.what == PowerStateController.POWER_STATE_LIVE_CHARGE ? mLiveChargeState :
            message.what == PowerStateController.POWER_STATE_SLEEP_CHARGE ? mSleepChargeState :
            message.what == PowerStateController.POWER_STATE_LIVE_UPDATE ? mLiveUpdateState :
            message.what == PowerStateController.POWER_STATE_SLEEP_UPDATE ? mSleepUpdateState :
            message.what == PowerStateController.POWER_STATE_CLOUD1 ? mCloud1State :
            message.what == PowerStateController.POWER_STATE_CLOUD2 ? mCloud2State :
            message.what == PowerStateController.POWER_STATE_MONITOR ? mMonitorState :
            // Fallback power state handling.
            /* message.what == PowerStateController.POWER_STATE_UNDEF ? */ mUndefState
            );
    }
}
private void transitionTo(State state) {
    // Exit old state.
    if ( mCurrState != null ) {
        Log.d( TAG, "Exiting state: " + mCurrState.getClass().getName() );
        mCurrState.exit();
    }
    mCurrState = state;
    // Enter new state.
    if ( mCurrState != null ) {
        Log.d( TAG, "Entering state: " + mCurrState.getClass().getName() );
        mCurrState.enter();
    }
}
//=================================================================
// Pre-defined Lucid Vehicle Power States
```

```
//=================================================================
private final UndefState mUndefState = new UndefState();
private final SleepState mSleepState = new SleepState();
private final WinkState mWinkState = new WinkState();
private final AccessoryState mAccessoryState = new AccessoryState();
private final DriveState mDriveState = new DriveState();
private final LiveChargeState mLiveChargeState = new LiveChargeState();
private final SleepChargeState mSleepChargeState = new SleepChargeState();
private final LiveUpdateState mLiveUpdateState = new LiveUpdateState();
private final SleepUpdateState mSleepUpdateState = new SleepUpdateState();
private final Cloud1State mCloud1State = new Cloud1State();
private final Cloud2State mCloud2State = new Cloud2State();
private final MonitorState mMonitorState = new MonitorState();
//=================================================================
// States definition
//=================================================================
// UndefState
class UndefState extends State {
    @Override void enter() {
        scheduleUndefProfileLogin();
    }
    @Override void exit() {
        cancelPendingUndefProfileLogin();
    }
}
private final Runnable mTimeOutVhalConnectTask = this::ensureUserProfileLogin; // 在下面关于用户登出登录的部分有定义
private static final int VHAL_CONNECTION_TIMEOUT = 10000;
private void scheduleUndefProfileLogin() {
    mHandler.postDelayed( mTimeOutVhalConnectTask, VHAL_CONNECTION_TIMEOUT );
}
private void cancelPendingUndefProfileLogin() {
    mHandler.removeCallbacks( mTimeOutVhalConnectTask );
}
 // SleepState
class SleepState extends State {
    @Override void enter() {
        scheduleSuspendToRam();
    }
    @Override void exit() {
        cancelPendingSuspendToRam();
    }
}
 // WinkState
class WinkState extends State {
    @Override void enter() {
        mOtaController.sendInstallNowIfCountDownStarted();
        ensureUserProfileLogout();
        setCccDisplaysOff();
    }
    @Override void exit() {  }
}
// AccessoryState
class AccessoryState extends State {
    @Override void enter() {
        sendOtaPowerStateBroadcast(OtaPowerStateConstants.POWER_STATE_ACCESSORY_BROADCAST, OtaPowerStateConstants.POW
        ensureUserProfileLogin();
        setCccDisplaysOn();
    }
    @Override void exit() {
        sendOtaPowerStateBroadcast(OtaPowerStateConstants.POWER_STATE_ACCESSORY_BROADCAST, OtaPowerStateConstants.POW
    }
}
// DriveState
class DriveState extends State {
    @Override void enter() {
        ensureUserProfileLogin();
        setCccDisplaysOn();
    }
    @Override void exit() {  }
}
 // LiveChargeState
class LiveChargeState extends State {
    @Override void enter() {
        ensureUserProfileLogin();
        setCccDisplaysOn();
    }
    @Override void exit() {  }
}
 // SleepChargeState
class SleepChargeState extends State {
    @Override void enter() {
        mOtaController.sendInstallNowIfCountDownStarted();
        ensureUserProfileLogout();
        setCccDisplaysOff();
    }
    @Override void exit() {  }
}
 // LiveUpdateState
class LiveUpdateState extends State {
    @Override void enter() {
        Log.i(TAG, "Inside PowerStateMachine...Power state : LiveUpdateState enter()");
        ensureUserProfileLogout();
        sendOtaPowerStateBroadcast(OtaPowerStateConstants.POWER_STATE_LIVE_UPDATE_BROADCAST, OtaPowerStateConstants.P
        setCccDisplaysOn();
    }
    @Override void exit() {
        Log.i(TAG, "Inside PowerStateMachine...Power state : LiveUpdateState exit()");
        sendOtaPowerStateBroadcast(OtaPowerStateConstants.POWER_STATE_LIVE_UPDATE_BROADCAST, OtaPowerStateConstants.P
    }
}
 // SleepUpdateState
class SleepUpdateState extends State {
    @Override void enter() {
        Log.i(TAG, "Inside PowerStateMachine...Power state : SleepUpdateState enter()");
        ensureUserProfileLogout();
        setCccDisplaysOff();
    }
    @Override void exit() {
        Log.i(TAG, "Inside PowerStateMachine...Power state : SleepUpdateState exit()");
    }
}
 // Cloud1State
class Cloud1State extends State {
    @Override void enter() {
        ensureUserProfileLogout();
        scheduleSuspendToRam();
    }
    @Override void exit() {
        cancelPendingSuspendToRam();
    }
}
 // Cloud2State
class Cloud2State extends State {
    @Override void enter() {
        ensureUserProfileLogout();
        scheduleSuspendToRam();
    }
    @Override exit() {
        cancelPendingSuspendToRam();
    }
}
 // MonitorState
class MonitorState extends State {
    @Override void enter() {
        ensureUserProfileLogout();
        setCccDisplaysOff();
    }
    @Override void exit() {}
}
```

- 剩余部分：基本上所有的内容都在

```
public class PowerStateMachine {
    private static final String TAG = "PowerStateMachine";
    private Context mContext;
    //=================================================================
    // Life cycle
    //=================================================================
    PowerStateController mPowerStateController;
    private OtaController mOtaController;
    private TcuWifiInterface mTcuWifiController;

    public PowerStateMachine() {
        mPowerStateController = new PowerStateController(); // 最主要就是它的调度器
    }
    public void init(Context context) {
        mContext = context;
        mPowerStateController.init( context, mPowerStateListener );
        mOtaController = new OtaController();
        mOtaController.init( context, null );
        // Init state machine.
        transitionTo( mUndefState ); // 最起始未定义状态
    }
    public void cleanup() {
        // Shut down state machine.
        transitionTo( mUndefState );
        mPowerStateController.cleanup();
        mOtaController.cleanup();
        mOtaController = null;
        mPowerStateController = null;
        mContext = null;
    }
    //=================================================================
    // PowerState Listener Callbacks
    //=================================================================
    private PowerStateController.PowerStateListener mPowerStateListener =
        new PowerStateController.PowerStateListener() { // public static class PowerStateListener 也是这么 new 出来的？
        @Override public void onVHalConnectedW() {  }
        @Override public void onVHalDisconnectedW() {  }
        @Override void onUpdatePowerStateW(int powerState) { // no public ?
            Log.d( TAG, "onUpdatePowerStateW(): powerState=" + powerState );
            if (powerState >= PowerStateController.POWER_STATE_SLEEP &&
                powerState <= PowerStateController.POWER_STATE_MONITOR) { // 只要是有效状态
                mHandler.sendEmptyMessage( powerState ); //
            } else {
                Log.w( TAG, "Unexpected powerState : " + powerState );
            }
        }
    };
    //=================================================================
    // State machine: 提取出去了，只留下面两行
    //=================================================================
    private final Runnable mTimeOutVhalConnectTask = this::ensureUserProfileLogin; // 就在下面关于用户登出登录的部分有定义
    private static final int VHAL_CONNECTION_TIMEOUT = 10000;
    //=========================================================================
    // Helper function to ensure if there is any user profile logged in and out: 发出要求用户切换的自定义广播
    //=========================================================================
     // It notifies LucidUserProfileApp to login into appropriate user-profile
    private void ensureUserProfileLogin() { // 要求首先登录进系统用户
        Log.d(TAG,"ensureUserProfileLogin()");
        // Notify LucidUserProfileApp to login to default user-profile
        Objects.requireNonNull(mContext).sendStickyBroadcastAsUser(new Intent(Utils.ACTION_LOGIN_TO_USER_PROFILE), UserHand
    }
     // *  It notifies LucidUserProfileApp to logout and switch back to system user
    private void ensureUserProfileLogout() { // 登出真正用户，回到系统用户
        Log.d(TAG,"ensureUserProfileLogout()");
        Objects.requireNonNull(mContext).sendStickyBroadcastAsUser(new Intent(Utils.ACTION_LOGOUT_FROM_USER_PROFILE), UserH
        stopWifiScan();
    }
    //=================================================================
    // Manage CCC Power States
    //=================================================================
    private void setCccDisplaysOn() {
        mPowerStateController.setCCCPowerState( PowerStateController.CCC_POWER_STATE_DISPLAYS_ON );
    }
    private void setCccDisplaysOff() {
        mPowerStateController.setCCCPowerState( PowerStateController.CCC_POWER_STATE_DISPLAYS_OFF );
    }
    private void setCccSuspendToRam() {
        mPowerStateController.setCCCPowerState( PowerStateController.CCC_POWER_STATE_SUSPEND_TO_RAM );
    }
    private final Runnable mTimeOutSuspendToRamTask = this::setCccSuspendToRam; // 就是上面定义的这个方法吗？
    private final static int SUSPEND_TO_RAM_DELAY = 2000; // 2 秒
    private void scheduleSuspendToRam() {
        mHandler.postDelayed( mTimeOutSuspendToRamTask, SUSPEND_TO_RAM_DELAY );
```

```java
    }
    private void cancelPendingSuspendToRam() {
        mHandler.removeCallbacks( mTimeOutSuspendToRamTask );
    }
    //================================================================
    public void sendOtaPowerStateBroadcast(int powerState, int startOrStop) { // 这里发的是 OTA 的广播？
        Intent intent = new Intent();
        intent.setPackage(OtaPowerStateConstants.OtaPackageName);
        intent.setAction(OtaPowerStateConstants.POWER_STATE_BROADCAST);
        intent.putExtra(OtaPowerStateConstants.POWER_STATE_TYPE_KEY, powerState);
        intent.putExtra(OtaPowerStateConstants.POWER_STATE_START_STOP_KEY, startOrStop);
        mContext.sendBroadcast(intent);
    }
    //=================================================================================
    // Stop TCU wifi scanning upon user logout: 这里不作重点，扫描无线网的作用目的是什么呢？
    //=================================================================================
    public TcuWifiInterface.TcuWifiListener mTcuWifiListener = new TcuWifiInterface.TcuWifiListener() {
        @Override public void onVhalConnectedW() {  }
        @Override public void onVhalDisconnectedW() {  }
        @Override public void onPropertiesReadyW() {
            if (mTcuWifiController != null) {
                mTcuWifiController.stopScan();
                mTcuWifiController.cleanup();
                mTcuWifiController = null;
            }
        }
        @Override public void onWifiEnabledW(boolean enabled) {}
        @Override public void onUpdateConnectionStatusW(TcuWifiController.ConnectionStatus connectionStatus) {}
        @Override public void onConnectionErrorW(TcuWifiController.ConnectionERROR error) {  }
        @Override public void onUpdateScanResultsW(List<WifiScanResult> result) {}
        @Override public void onNewScanResultsW() {}
        @Override public void onWifiConnectedW(WifiScanResult connectedEntry) {}
        @Override public void onConnectingSavedNetwork() {}
    };
    private void stopWifiScan() {
        mTcuWifiController = SystemWifiController.systemController(mContext, mTcuWifiListener, true);
    }
}
```

3 LucidCarSettings 应用：启动过程与及全局配置相关

3.1 AndroidManifest.xml

```xml
  <receiver
      android:name="com.lucid.car.carsettings.BootReceiver"
      android:exported="true">
    <intent-filter>
      <action android:name="android.intent.action.BOOT_COMPLETED" />
    </intent-filter>
  </receiver>
  <receiver
      android:name="com.lucid.carlib.profile.ProfileSwitcherService$Receiver"
      android:permission="com.lucid.permission.RECEIVE_USERPROFILE_INTENTS">
    <intent-filter>
      <action android:name="com.lucid.intent.action.USER_LOGGED_IN" />
      <action android:name="com.lucid.intent.action.USER_LOGGED_OUT" />
    </intent-filter>
  </receiver>
  <service
      android:name="com.lucid.carlib.profile.ProfileSwitcherService"
      android:exported="false" />
  <meta-data
      android:name="ProfileSwitchHandlerClass"
      android:value="com.lucid.car.carsettings.profileAndAccess.SettingsProfileSwitcher" />

<!-- Content Provider -->
  <provider
      android:name=".connectivity.homelink.Room.provider.HomeLinkProvider"
      android:authorities="com.lucid.car.carsettings.connectivity.homelink.Room.provider"
      android:exported="true" />
  <!-- IPC AIDL 进程间服务 -->
  <service
```

```xml
        android:name="com.lucid.car.carsettings.profileAndAccess.LoginService"
        android:enabled="true"
        android:exported="true"
        android:launchMode="singleInstance">
      <intent-filter>
        <category android:name="android.intent.category.DEFAULT" />
      </intent-filter>
    </service>
  <!-- 起始 -->
    <activity
        android:name=".LaunchWrapperActivity"
        android:theme="@style/transparentTheme"
        android:exported="true">
      <!-- Set priority high enough to trump the phone and aosp car setting app -->
      <!-- TODO: once phone setting and aosp car setting is removed from car system image, set priority to 1 -->
      <intent-filter android:priority="11">
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
      </intent-filter>
    </activity>
    <activity
        android:name=".home.HomeActivity"
        android:allowEmbedded="true"
        android:launchMode="singleTask"
        android:windowSoftInputMode="stateUnspecified|adjustResize"
        android:resizeableActivity="true">
      <intent-filter android:priority="11">
        <action android:name="com.lucid.car.carsettings" />
        <category android:name="android.intent.category.DEFAULT" />
      </intent-filter>
    </activity>
    <!-- SettingsApp 需要处理的各种配置与广播接收 -->
    <activity
        android:name=".connectivity.BluetoothPairingDialog"
        android:excludeFromRecents="true"
        android:theme="@style/AppDialogTheme"
        android:windowSoftInputMode="stateVisible|adjustResize"
        android:allowEmbedded="true"
        android:launchMode="singleTask"
        android:resizeableActivity="true" />
    <service android:name=".connectivity.BluetoothPairingService" />
    <receiver android:name=".alexa.AlexaAuthStateReceiver">
      <intent-filter>
        <action android:name="com.lucid.assistant.ACTION_ALEXA_AUTHORIZE" />
        <action android:name="com.lucid.assistant.ACTION_ALEXA_NETWORK_CONNECTED" />
      </intent-filter>
    </receiver>
  <!-- 系统重置恢复为厂配 -->
    <activity
        android:name=".profileAndAccess.factoryReset.FactoryResetCidDisplay"
        android:launchMode="singleTask"
        android:windowSoftInputMode="stateUnspecified|adjustResize"
        android:resizeableActivity="true"/>
    <activity
        android:name=".profileAndAccess.factoryReset.FactoryResetProgressActivity"
        android:exported="false"
        android:launchMode="singleTask"
        android:windowSoftInputMode="stateUnspecified|adjustResize"
        android:resizeableActivity="true"/>

    <!-- 用户注册/用记注册完成相关的服务与广播接收器 -->
    <service
        android:name=".fte.FTEService"
        android:enabled="true">
      <intent-filter>
        <category android:name="android.intent.category.DEFAULT" />
      </intent-filter>
    </service>
    <receiver android:name=".fte.FTEReceiver">
      <intent-filter>
        <category android:name="android.intent.category.DEFAULT" />
        <action android:name="com.lucid.userprofile.ACTION_NEW_PROFILE_CREATED" />
      </intent-filter>
    </receiver>
    <receiver
```

```
      android:name=".profileAndAccess.AccessAndProfileFragment$FTECompleteReceiver"
      android:exported="true">
    <intent-filter>
      <category android:name="android.intent.category.DEFAULT" />
    </intent-filter>
    <intent-filter>
      <action android:name="com.lucid.car.userprofile.action.FTE_COMPLETE" />
    </intent-filter>
  </receiver>
  <!-- 语言相关 -->
  <service android:name=".languagePackage.LanguagePackageDownloadService"
           android:enabled="true">
    <intent-filter>
      <category android:name="android.intent.category.DEFAULT" />
    </intent-filter>
  </service>
```

3.2 BootReceiver.java: 能够在系统起来之后，需要第一时间处理的事情，比如向下传信号

```
public class BootReceiver extends BroadcastReceiver {
@Override
    public void onReceive(Context context, Intent intent) {
    String action = intent.getAction();
    if (action != null) {
        if (action.equals(Intent.ACTION_BOOT_COMPLETED)) {
            Log.e(TAG, "BOOT COMPLETED CALLED FROM CAR SETTINGS");

            // Settings.Global.putInt(context.getContentResolver(), WIFI_SCAN_RUNNING, 0);
            int scanRunning = Settings.Global.getInt(context.getContentResolver(), WIFI_SCAN_RUNNING, 0);
            Log.d(TAG, "......... scanRunning: " + scanRunning);

            Settings.Global.putString(context.getContentResolver(), OTA_COUNT_DOWN_TIMER_STARTED, OTA_COUNT_DOWN_TIMER_STAR
            Log.d(TAG, "OTA_COUNT_DOWN_TIMER_STARTED value"
                  + Settings.Global.getString(context.getContentResolver(), OTA_COUNT_DOWN_TIMER_STARTED));

            if (!Process.myUserHandle().equals(UserHandle.SYSTEM)) { // 只要不是系统用户，那么就需要索要用户数据信息
                onRebootSendTripInfo(context);
                Log.d(TAG, "  logged in user, disable aosp settings BT pairing dialog");
                PackageManager pm = context.getPackageManager();
// PackageManager 提供了一个方法，setComponentEnabledSetting()，这个方法的作用是启用或者禁用四大组件
// 这里启用的是像任何手机安卓系统一样，系统自带的系统级别的 Settings 应用中的蓝牙相关的自动配对模块相关的信息
                pm.setComponentEnabledSetting(new ComponentName("com.android.car.settings",
                                                "com.android.car.settings.bluetooth.BluetoothPairingRequest
// 比如我们想禁用一个服务，就可以使用下面的方法，传入的参数就是服务的名称
                                              PackageManager.COMPONENT_ENABLED_STATE_DISABLED, PackageManager.DONT_KILL_APP
// 如果想重新启用的话，也是上面的方法，只是 setComponentEnabledSetting() 第二个参数值不一样，
// 只需要用 PackageManager.COMPONENT_ENABLED_STATE_ENABLED 来替换 PackageManager.COMPONENT_ENABLED_STATE_DISABLED 即可
                    }

            // send data sharing signal
            int dataSharing = Settings.Global.getInt( context.getContentResolver(), DATA_SHARING_ENABLED, 1 );
            boolean dataSharingEnabled = (dataSharing == 1 ) ? true : false;
            Log.d(TAG, " dataSharing: " + dataSharing + " dataSharingEnabled: " + dataSharingEnabled);
            TcuDataController tcuDataController = new TcuDataController();
            tcuDataController.init(context, mTcuDataListener);
            // send data sharing  request to tcu
            tcuDataController.setDataSharingEnabled(dataSharingEnabled);
        }
    }
}

private void onRebootSendTripInfo(Context context) {
    final BcmController mBcmController = new BcmController();
    Log.d(TAG, "onRebootSendTripInfo " + mBcmController.toString());
    SharedPreferences mSharedPreference = context.getSharedPreferences(HomeActivity.PREF_SETTINGS_APP, Context.MODE_PRIVATE
    int tripInfoValue = mSharedPreference != null ? mSharedPreference.getInt(PREF_TRIP_INFO, 0) : 0;
    Log.d(TAG, "onRebootSendTripInfo trip info " + tripInfoValue);
    mBcmController.init(context, new BcmController.BcmListener() {
            @Override
                public void onPropertiesReadyW() {
                super.onPropertiesReadyW();
                Log.d(TAG, "onRebootSendTripInfo  bcm controller onPropertiesReadyW " + mBcmController + " trip info  " + t
                if (mBcmController != null)
                    mBcmController.setTripInfoDisplay(tripInfoValue);
            }
        });
}
private final TcuDataController.TcuDataListener mTcuDataListener = new TcuDataController.TcuDataListener() {  };
private void startLanguagePackDownloadService(Context context) {
    Log.d(TAG, "startLanguagePackDownloadService");
    Intent serviceIntent = new Intent(context, LanguagePackageDownloadService.class);
    context.startService(serviceIntent);
}
}
```

3.3 AsyncResources.java: 弄个简洁片的出来看下，好像没什么东西

```
public class AsyncResources {
    private static final String TAG = "AsyncResources";
//================================================================
// Static Singleton APIs
//================================================================
    private static AsyncResources sInstance = null;
    public static synchronized AsyncResources getInstance(Context context) {
        if (sInstance == null)
            sInstance = new AsyncResources(context);
        return sInstance;
    }
//================================================================
// Constructor
//================================================================
    private final Context mContext;
    private final ProfileData mProfileData;
    // private final BcmController mBcmController; // 关于远光灯自适应，是有可能会用到这个东西的
    private final UnitPrefController mUnitPrefController;
    private boolean mInitControllers = false;

    private AsyncResources(Context context) {
        mContext = context.getApplicationContext();
        mProfileData = new ProfileData(context);
        mUnitPrefController = new UnitPrefController();
    }
//================================================================
// Connections Management APIs
//================================================================
    public static abstract class ConnectionsListener {
        ProfileData.ServiceConnectionListener mProfileListener;
        UnitPrefController.UnitPrefListener mUnitPrefListener;

        public ConnectionsListener(ProfileData.ServiceConnectionListener profileListener,
                                   UnitPrefController.UnitPrefListener unitPrefListener,
            ) {
            mProfileListener = profileListener;
            mUnitPrefListener = unitPrefListener;
        }
        public abstract void onConnectedAll();
        public abstract void onDisconnectedAny();
    }
    private final List<ConnectionsListener> mConnectionsListeners = new ArrayList<>();
    public void registerConnection(ConnectionsListener listener) {
        if (listener == null) {
            Log.w(TAG, "registerConnection() : ConnectionListener is null");
            return;
        }
        if (isReadyProfileData() && listener.mProfileListener != null)
            listener.mProfileListener.onConnected();
        if (isReadyUnitPrefController() && listener.mUnitPrefListener != null)
            listener.mUnitPrefListener.onVhalConnectedW();
        if (isReadyAll())
            listener.onConnectedAll();
        synchronized (mConnectionsListeners) {
            /*  Optimize to only initialize once. */
            if ( !mInitControllers ) {
                Log.d( TAG, "=== Initializing controllers." );
                mInitControllers = true;
                mProfileData.bindService(mProfileDataListener);
                mUnitPrefController.init(mContext, mUnitPrefControllerListener);
            }
            mConnectionsListeners.add(listener);
        }
    }
    public void unregisterConnections(ConnectionsListener listener) {
        if (listener == null) {
            Log.w(TAG, "unregisterConnections() : ConnectionListener is null");
            return;
        }
        if (isReadyAll())
            listener.onDisconnectedAny();
        if (isReadyProfileData() && listener.mProfileListener != null)
            listener.mProfileListener.onDisconnected();
        if (isReadyUnitPrefController() && listener.mUnitPrefListener != null)
            listener.mUnitPrefListener.onVhalDisconnectedW();
        synchronized (mConnectionsListeners) {
            mConnectionsListeners.remove(listener);
            // Optimize to never clean up.
            /* if (mConnectionsListeners.isEmpty()) {
               mProfileData.unbindService();
               mBcmController.cleanup();
               mAdasController.cleanup();
               mUnitPrefController.cleanup();
               mGearController.cleanup();
               mDriveModeController.cleanup();
               } */
        }
    }
    public boolean isReadyAll() {
        return (isReadyProfileData()
                && isReadyUnitPrefController());
    }
    public boolean isReadyProfileData() {
        return (mProfileData != null && mProfileData.isServiceBound());
    }
    public boolean isReadyUnitPrefController() {
        return (mUnitPrefController != null && mUnitPrefControllerReady);
    }
//=================================================================
// Resource Getter APIs
//=================================================================
    public ProfileData getProfileData() {
        return mProfileData;
    }
    public UnitPrefController getUnitPrefController() {
        return mUnitPrefController;
    }
//=================================================================
// Listener Wrappers
//=================================================================
    private boolean mUnitPrefControllerReady = false;
    private void checkConnectedAll() {
        if (isReadyAll()) {
            synchronized (mConnectionsListeners) {
                for (ConnectionsListener listener : mConnectionsListeners)
                    listener.onConnectedAll();
            }
        }
    }
    private void checkDisconnectedAny() {
        if (!isReadyAll()) {
            synchronized (mConnectionsListeners) {
                for (ConnectionsListener listener : mConnectionsListeners)
                    listener.onDisconnectedAny();
            }
        }
    }
    /**
     * mProfileDataListener wrapper
     */
    private final ProfileData.ServiceConnectionListener mProfileDataListener =
        new ProfileData.ServiceConnectionListener() {
        public void onConnected() {
            synchronized (mConnectionsListeners) {
```

```java
            for (ConnectionsListener listener : mConnectionsListeners) {
                if (listener.mProfileListener != null) {
                    listener.mProfileListener.onConnected();
                    int level = mProfileData.retrieveInt(ProfileKey.SETTINGS_ADAS_LDP_LEVEL,
                                                         AdasPrefs.getLaneDepartureLevelDefault());
                    mAdasController.setInitialLdpValue(level);
                }
            }
        }
        checkConnectedAll();
    }
    public void onDisconnected() {
        checkDisconnectedAny();
        synchronized (mConnectionsListeners) {
            for (ConnectionsListener listener : mConnectionsListeners) {
                if (listener.mProfileListener != null)
                    listener.mProfileListener.onDisconnected();
            }
        }
    }
};
/**
 * UnitPrefController wrapper
 */
private final UnitPrefController.UnitPrefListener mUnitPrefControllerListener =
    new UnitPrefController.UnitPrefListener() {
    @Override
    public void onVhalConnectedW() {
        super.onVhalConnectedW();
        synchronized (mConnectionsListeners) {
            for (ConnectionsListener listener : mConnectionsListeners) {
                if (listener.mUnitPrefListener != null)
                    listener.mUnitPrefListener.onVhalConnectedW();
            }
        }
        mUnitPrefControllerReady = true;
        checkConnectedAll();
    }
    @Override
    public void onVhalDisconnectedW() {
        super.onVhalDisconnectedW();
        mUnitPrefControllerReady = false;
        checkDisconnectedAny();
        synchronized (mConnectionsListeners) {
            for (ConnectionsListener listener : mConnectionsListeners) {
                if (listener.mUnitPrefListener != null)
                    listener.mUnitPrefListener.onVhalDisconnectedW();
            }
        }
    }
};
/**
 * mBcmControllerListener wrapper
 */
public BcmController.BcmListener getBcmListener() {
    return mBcmControllerListener;
}
private final BcmController.BcmListener mBcmControllerListener =
    new BcmController.BcmListener() {
    @Override
    public void onUpdateHighBeamPowerW(boolean isOn) {
        synchronized (mConnectionsListeners) {
            for (ConnectionsListener listener : mConnectionsListeners) {
                if (listener.mBcmListener != null)
                    listener.mBcmListener.onUpdateHighBeamPowerW(isOn);
            }
        }
    }
    @Override
    public void onUpdateHighBeamSensitivityW(int val) {
        synchronized (mConnectionsListeners) {
            for (ConnectionsListener listener : mConnectionsListeners) {
                if (listener.mBcmListener != null)
                    listener.mBcmListener.onUpdateHighBeamSensitivityW(val);
            }
```

```
                }
            }
            @Override
            public void onUpdateLSCModeStatusW(int status) {
                synchronized (mConnectionsListeners) {
                    for (ConnectionsListener listener : mConnectionsListeners) {
                        if (listener.mBcmListener != null)
                            listener.mBcmListener.onUpdateLSCModeStatusW(status);
                    }
                }
            }
        };
}
```

4 LucidSystemUI: 每天回来看，理解都再深入一点儿

- 这个应用监听 BOOT_COMPLETE 广播，收到广播信息后，同时开启多个服务

4.1 AndroidManifest.xml

```
<application
    android:allowBackup="true"
    android:icon="@mipmap/ic_launcher"
    android:label="@string/app_name"
    android:roundIcon="@mipmap/ic_launcher_round"
^^Iandroid:supportsRtl="true" <!-- 支持从右向左显示 -->
    tools:replace="android:appComponentFactory"
    android:appComponentFactory="lucid">
    <!-- 这里是可能会出现很多的 compiler error 的: 用自己配置的 lucid 包 -->
    <!-- 这里涉及到很多根据车 (车门开关/用户的登录状态, Gear status) 的状态来决定 CID 的状态等 -->
  <activity
      android:name="com.lucid.car.sysui.pinToDrive.PinToDriveActivity"
      android:label="@string/pin_to_drive_label"
      android:exported="true"
      android:theme="@android:style/Theme.Translucent.NoTitleBar">
  </activity>
  <service
      android:name="com.lucid.car.sysui.LucidSysUiService"
      android:enabled="true"
      android:launchMode="singleInstance"
      android:directBootAware="true"
      android:exported="true">
    <intent-filter> <!-- 下面这个是安卓 车载 automotive 系统广播 -->
      <action android:name="android.car.settings.ADD_ACCOUNT_SETTINGS" />
      <category android:name="android.intent.category.DEFAULT" />
    </intent-filter>
  </service>
  <!-- !!! Only used for px3auto dev environment !!! -->
  <activity
      android:name="com.lucid.car.sysui.LaunchCidActivity"
      android:icon="@mipmap/ic_launcher"
      android:label="@string/launch_cid"
      android:launchMode="singleInstance"
      android:taskAffinity=".LaunchCidActivity" >
    <intent-filter>
      <action android:name="android.intent.action.MAIN" />
      <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
  </activity>

  <service android:name="com.lucid.car.sysui.profiles.ProfileManagerService"/>
  <receiver android:name=".LucidSysUiService$NotificationReceiver"/>

  <service android:name=".pinToDrive.PinToDriveService"
           android:enabled="true"
           android:launchMode="singleInstance"
           android:exported="true"/>

  <receiver android:name=".pinToDrive.PinToDriveService$PinToDriveBootReceiver">
    <intent-filter>
      <action android:name="android.intent.action.BOOT_COMPLETED"/>
    </intent-filter>
  </receiver>

  <receiver android:name=".LucidSysUiService$BootReceiver" >
    <intent-filter>
      <action android:name="android.intent.action.BOOT_COMPLETED"/>
    </intent-filter>
  </receiver>

  <receiver android:name=".notification.ICRCIDCommunicationReceiver"
            android:permission="com.lucid.car.sysui.PERMISSION_ICR_CID_COMMUNICATION">
    <intent-filter>
      <action android:name="com.lucid.car.sysui.ACTION_ICR_CID_COMMUNICATION"/>
      <category android:name="android.intent.category.DEFAULT" />
    </intent-filter>
  </receiver>

  <receiver android:name=".notification.HomeLinkNotificationReceiver"
            android:permission="com.lucid.car.sysui.PERMISSION_ICR_CID_HOME_LINK_COMMUNICATION">
    <intent-filter>
      <action android:name="com.lucid.car.sysui.ACTION_ICR_CID_HOME_LINK_PROGRAM_NOTIFICATION" />
      <category android:name="android.intent.category.DEFAULT" />
    </intent-filter>
  </receiver>
</application>
```

4.2 LucidSysUiService.java

- 监听 BOOT_COMPLETE 广播，收到广播信息后 (问题是，如果应用已经死了，它还能接收到这个系统广播吗???)，为主人家罗列出常用路线等。。。?????

```
<service
    android:name="com.lucid.car.sysui.LucidSysUiService"
    android:enabled="true"
    android:launchMode="singleInstance"
    android:directBootAware="true" <!-- 在锁屏的状态下是可以启动的 -->
    android:exported="true"> <!-- 是否支持其它应用调用当前组件 -->
    <intent-filter>
      <action android:name="android.car.settings.ADD_ACCOUNT_SETTINGS" /> <!-- 还需要追一下这个动作做了什么??? -->
    <category android:name="android.intent.category.DEFAULT" />
  </intent-filter>
</service>
```

- 对于系统广播，由于是系统内部直接发出，无法更改此 intent flag 值，因此，3.1 开始对于静态注册的接收系统广播的 BroadcastReceiver，如果 App 进程已经退出，将不能接收到广播。(这句话是什么意思呢？)
- 但是对于自定义的广播，可以通过复写此 flag 为 FLAG_INCLUDE_STOPPED_PACKAGES，使得静态注册的 BroadcastReceiver，即使所在 App 进程已经退出，也能能接收到广播，并会启动应用进程，但此时的 BroadcastReceiver 是重新新建的。(仅适用于自定义广播，系统广播仍然是不可以的)
- 启动方式
  - start 方式启动：onCreate() onStartCommand() stopService onDestroy()
  - bind 方式启动：onCreate() onBind() unbindService onUnbind() onDestroy()
- 如果当前 app 的 uid 不是 active（处于后台），即使启动的是前台 Service，也可能会失败，例如：

```
ActivityManager: Background start not allowed: service Intent { act=geofence_trigered cmp=com.xiaomi.smarthome/.scene.act
```

- 原理代码如下，下面的就是上面 1 中列出的代码：

```
final boolean bgLaunch = !mAm.isUidActiveLocked(r.appInfo.uid); // 当前进程 LucidCarService App, 或当前服务 LucidSystemServic
boolean forcedStandby = false;
if (bgLaunch && appRestrictedAnyInBackground(r.appInfo.uid, r.packageName)) {
    forcedStandby = true;
}
```

- 被调用的前台服务需要有前台服务的权限，这个已经有了
- 在这里，就是想知道，LucidSystemService 想要启动 LucidSysUiService 的时候，它到底是前台服务，还是它自己当时正在后台服务，因此不能成功启动 LucidSysUiService?

```
<receiver android:name=".LucidSysUiService$BootReceiver" >
    <intent-filter>
        <action android:name="android.intent.action.BOOT_COMPLETED"/>
    </intent-filter>
</receiver>
```

- 我们希望 LucidSysUiService.java 能够长存于系统中，希望它是活的；可是如果它已经死了，它还可以接收到系统所发出的 BOOT_COMPLETED 广播吗?
- 是因为它是死的，还是因为上次，也就是在 reboot 之前没有 clean 干净，以致导致了 broken NotificationChannel 呢？需要两种可能的原因都有办法以能够确认一下
  - 创建和管理通知渠道，从 Android8.0 开始，需要为发送的每种不同类型的通知创建一个渠道，如果在 Android8.0 及以上在未指定通知频道的情况下发送通知，通知不会显示，会记录错误；
  - 重写了 NotificationChannel 的创建方法以及 clean up, almost the same as PinToDrive, will it be BETTER?
  - LucidCarService APP:
- When to destroy LucidSysUiService?

```
public class LucidSysUiService extends Service {
    // !!! HACK !!! Storing static singleton instance for HomeLink initialization.
    private static LucidSysUiService S_INSTANCE = null;
    public static class BootReceiver extends BroadcastReceiver { // 静态的
        @Override
        public void onReceive(Context context, Intent intent) {
            if ( Process.myUserHandle().equals( UserHandle.SYSTEM ) ) {
                Log.d(TAG, "Exiting user0");
                return;
            }
            if (Intent.ACTION_BOOT_COMPLETED.equals(intent.getAction())) { // 收听到 BOOT_COMPLETED 广播
                Log.d(TAG, "Boot up completed" + intent.getAction());
                // OTA user consent
                createAlarmManager(context, false);
                // Homelink initialization
                if ( S_INSTANCE != null ) { // 有没有什么异常情况下它是空的呢？
                    S_INSTANCE.mIcrSettingsWindow.getHomeLinks();
                }
            }
        }
    }
    private void registerICRCIDHomeLinkCommunicationBroadcast(){
        IntentFilter intentFilter = new IntentFilter();
        intentFilter.addAction(ACTION_ICR_CID_HOME_LINK_PROGRAM_NOTIFICATION);
        mLocalBroadcastManager.registerReceiver(mICRCIDHomeLinkCommunicationReceiver, intentFilter);
    }
    private final BroadcastReceiver mICRCIDHomeLinkCommunicationReceiver = new BroadcastReceiver() {
        @Override
        public void onReceive(Context context, Intent intent) {
            Log.d(TAG, "onReceive");
            if (intent != null && ACTION_ICR_CID_HOME_LINK_PROGRAM_NOTIFICATION.equals(intent.getAction())) {
                String type = intent.getStringExtra(HomeLinkNotificationReceiver.EXTRA_ICR_CID_HOME_LINK_COMMUNICATION_TYPE
                if (HomeLinkNotificationReceiver.ICR_CID_COMMUNICATION_TYPE_HOME_LINK_PROGRAMMED.equals(type)) {
                    Log.d(TAG, "getHomelink");
                    mIcrSettingsWindow.getHomeLinks(); // 这里从 ContentProvider 数据库提取更新后的数据, 在 IcrSettingsWindow
                    // 这里好像理解偏了: homelinks 链表是与家里车库等的联接, 涉及蓝牙, 不理解暂时关系不是很大
                }
            }
        }
    };
    private void registerICRCIDCommunicationBroadcast() {
        IntentFilter intentFilter = new IntentFilter();
        intentFilter.addAction(ACTION_ICR_CID_COMMUNICATION);
        mLocalBroadcastManager.registerReceiver(mICRCIDCommunicationReceiver, intentFilter);
    }
    // 这个东西, 注册一场, 就只为 BugReport 吗?感觉这里的部分逻辑还没有联接起来
    private BroadcastReceiver mICRCIDCommunicationReceiver = new BroadcastReceiver() {
        @Override
        public void onReceive(Context context, Intent intent) {
            if(mBugReportNotificationWindow.isBugReportEvent(intent)) {
                // 这个窗口, 改天需要找出一个这样的窗口看一下, 暂时不追了。。。
                mBugReportNotificationWindow.show(intent);
            }
        }
    };
}
```

4.3 IcrSettingsWindow.getHomelinks(): 小主想去哪里呢?现有以下常项供您选择。。。

- 应该是车启动想走路, 那么聪明的车载系统就会帮小主人想好: 小主想要去哪里呢? 现在以下常项可供选择???????。。。。。。。
- 没读懂没连贯起来, 改天再补这个

```
public void getHomeLinks() {
    // Get saved homelink programmed channels
    Cursor cursor;
    cursor = mContext.getContentResolver().query( Uri.parse(CONTENT_URI), null, null, null, "homeLink_id");
    Log.d(TAG, "cursor data" +cursor);
    if (cursor != null) {
        if (homeLinkList != null) {
            homeLinkList.clear();
        }
        if (homeLinkListInNearByRange != null) {
            homeLinkListInNearByRange.clear();
            mIsHomeLinkInRangeInit = false;
        }
        if (cursor.getCount() > 0) {
            while (cursor.moveToNext()) {
                homeLinkList.add (new HomeLink(cursor.getInt(0),
                                               cursor.getString(1), cursor.getFloat(2),
                                               cursor.getFloat(3), cursor.getInt(4)));
            }
        }
        if (homeLinkList != null)
            Log.d(TAG, "homeLinkList Size" + homeLinkList.size());
        cursor.close();
    }
}
```

4.4 ProfileGridAdapter 账户切换下接链表

```
public class ProfileGridAdapter extends RecyclerView.Adapter<ProfileGridAdapter.UserAdapterViewHolder>
    implements CarUserManagerHelper.OnUsersUpdateListener {
private static final String LUCID_ADMIN = "lucid admin";
private final CarUserManagerHelper mCarUserManagerHelper; // android aosp 系统类

private final Context mContext;
private List<UserInfo> userInfoList;
private final UserInfo currentUser;
private final ProfileGridCallback callback; // 回调
private final String TAG = "ProfileGridAdapter";

public interface ProfileGridCallback { // 定义一个公用的回调接口
```

```
    void onFinishEvent();
}

public ProfileGridAdapter(Context context, ProfileGridCallback callback) {
    mContext = context;
    this.callback = callback;
    mCarUserManagerHelper = new CarUserManagerHelper(mContext);
    userInfoList = mCarUserManagerHelper.getAllUsers();
    currentUser = mCarUserManagerHelper.getCurrentProcessUserInfo();
}

@NonNull @Override public UserAdapterViewHolder onCreateViewHolder(@NonNull ViewGroup viewGroup, int i) {
    LayoutInflater layoutInflater = LayoutInflater.from(viewGroup.getContext());
    View view = layoutInflater.inflate(R.layout.profile_grid_item, null);
    return new ProfileGridAdapter.UserAdapterViewHolder(view);
}
@Override public void onBindViewHolder(@NonNull ProfileGridAdapter.UserAdapterViewHolder viewHolder, int i) {
    UserInfo userInfo = userInfoList.get(i);
    Drawable icon = UserIcons.getDefaultUserIcon(mContext.getResources(), userInfo.id, /* light= */ true);
    viewHolder.mUserNameTextView.setText(userInfo.name);
    if (icon == null)
        viewHolder.mUserAvatarImageView.setImageResource(R.drawable.icn_general_profile_active);
    else
        viewHolder.mUserAvatarImageView.setImageDrawable(icon);
    boolean isActive = mCarUserManagerHelper.getCurrentForegroundUserId() == userInfo.id;
// 定义下拉链表的点击事件，申请转换用户，并设置回调
    viewHolder.mView.setOnClickListener(v -> {
            if (!isActive) {
                // 将当前索求的 foreground User request 写入数据库
                Settings.Global.putInt(mContext.getContentResolver(), ProfileKey.GLOBAL_SETTING_HWA_LOGIN_SIGNAL_WAS_SENT,
                Log.d(TAG, "Global.Settings " + ProfileKey.GLOBAL_SETTING_HWA_LOGIN_SIGNAL_WAS_SENT + "=0");
                // 并请安卓 10 多用户系统，根据用户 userId，帮助切换用户：
// * In addition, one more instance is spun up for each 2ndary user profile.
// * This instance's life cycle is tied to user profile login session,
// * and is triggered by listening to BOOT_COMPLETE broadcast intent.
                // 这个系统切换的背后，还有几座高山几汪海洋，你就慢慢去爬去游把它们搞懂!!!
                mCarUserManagerHelper.switchToUser(userInfo);

            } else // 当切换用户成功并完成，调用完成后的回调方法 (ProfileManagerService.java 会调用此回调方法)
                callback.onFinishEvent();
        });
}
```

4.5 ProfileManagerService.java：这个服务，死去活来，都不明不白。。。。。。

```
public class ProfileManagerService extends Service implements ProfileGridAdapter.ProfileGridCallback {
    private static final String TAG = "ProfileManagerService";

    public static final String USER_CHOSER_ACTION = "choose user UI";
    public static final String LOGIN_CHOSER_OPTION = "login choose option UI";

    @Override
        public void onFinishEvent() { // 覆写回调接口方法
        // todo need implement pin here or before call this: 应该 lucid 用了其它的方法绕过了这些，找出逻辑来
        navigateToHomeApp();
    }
    @Override // Service 的生命周期方法：是谁发意图启动这个服务的???
        // * in case we use sticky service we should stop service after use it：什么时候关的？用户切换的时候系统处理的？
        public int onStartCommand(@Nullable Intent intent, int flags, int startId) {
        Log.e(TAG, "onHandleIntent has received command: " + intent + ", Thread: " + Thread.currentThread());
        if (intent != null && intent.getAction() != null) {
            String action = intent.getAction();
            switch (action) {
            case LOGIN_CHOSER_OPTION:
                launchLoginChoserView();
                break;
            default:
                Log.w(TAG, "action is not been handled: " + action);
            }
        } else {
            Log.w(TAG, "intent is null or not action");
        }
        return START_STICKY;
    }
```

```java
    private void showWelcomeScreen(String successString, boolean isSuccess, @Nullable Drawable drawable) {
        // ......
        // set up timeout event
        if (isSuccess) {
            Handler handler = new Handler();
            int id = ProfileHelper.getInstance(getBaseContext()).getCurentuser().id;
            Drawable icon = UserIcons.getDefaultUserIcon(getResources(), id, /* light= */ true);
            handler.postDelayed(() -> showWelcomeScreen(getString(R.string.welcome_tx), false, icon), 1000);
        } else if (isWelcome) {
            Handler handler = new Handler();
            handler.postDelayed(this::navigateToHomeApp, 3000);
        } else {

        }
    }
    private void navigateToHomeApp() { // 所做的一切: 只是关闭当前窗口; 那么接下来显示的内容是什么呢？
        //todo add navigate to home app
        mWindowManager.removeViewImmediate(currentView);
        currentView = null;
        stopSelf();
    }
}
```

4.6 ProfileListAdapter.java: 真正的账户下拉链表, 部分涉及系统用户管理

```java
// http://hqdevbld01.corp.lucid.lcl:8888/source/xref/lucid_dash5dv_10r40/packages/services/Car/user/car-user-lib/src/androi
public final class CarUserManagerHelper {
    private static final String TAG = "CarUserManagerHelper";
/**
 * Interface for listeners that want to register for receiving updates to changes to the users
 * on the system including removing and adding users, and changing user info.
 */
    public interface OnUsersUpdateListener {
         // * Method that will get called when users list has been changed. ???????
        void onUsersUpdate();
    }
}
import android.car.userlib.CarUserManagerHelper; // 自家定义的
public class ProfileListAdapter extends RecyclerView.Adapter<ProfileListAdapter.UserAdapterViewHolder>
implements CarUserManagerHelper.OnUsersUpdateListener  {
    @Override
        public void onUsersUpdate() {
        clearUsers();
        notifyDataSetChanged(); // 通知链表数据改变
    }
    public void clearUsers() {
        userInfoList.clear(); // 清空本地 (ProfileListAdapter) 用户信息链表 (系统的当然不该变), 释放内存的意思吗？
    }
    public boolean isUserStopping() { // IcrSettingsWindow ProfilesTab 中用到的方法定义
        for ( UserInfo userInfo : mCarUserManagerHelper.getAllUsers() ) {
            if ( userInfo.id == 0 ) {
                continue;  // Skip user0.
            } else if ( userInfo.id == mCarUserManagerHelper.getCurrentForegroundUserId() ) {
                continue;  // Skip foreground user.
            } else if ( mUserManager.isUserRunningOrStopping( userInfo.getUserHandle() ) ) {
                // UserManager.isUserRunningOrStopping(userId): 系统方法, 判断是否是正在退出或是已经退出的前用户
                // Someone is still trying to logout.
                return true;
            }
        }
        return false;
    }
}
```

4.7 IcrSettingsWindow ProfilesTab

```java
public class IcrSettingsWindow implements LifecycleOwner, LocationListener {
    // ....
class ProfilesTab extends SettingsTab {
private final RecyclerView mProfilesList;
private final ProfileListAdapter mProfilesAdapter;
public static final String ACTION_CREATE_PROFILE = "com.lucid.car.settings.CREATE_PROFILE";
```

```
ProfilesTab() { // 前面有定义 SettingsTab basetab, 可以回去查
    super( mIcrSettingsWindow.findViewById(R.id.settings_tab_profiles),
           mIcrSettingsWindow.findViewById(R.id.profiles_drag_line),
           mIcrSettingsWindow.findViewById(R.id.settings_profiles) );
    mProfilesList = mTabView.findViewById(R.id.profiles_list_view);
    mDeepLinkSettings.setVisibility( View.VISIBLE );
    mDeepLinkSettings.setText( mContext.getResources().getString(R.string.settings_tab_profiles_profile_settings) );
    dividerLine.setVisibility( View.VISIBLE ); // Tab 下面标识当前 Tab 为激活状态的下划线

    mProfilesAdapter = new ProfileListAdapter(mContext);
    mProfilesAdapter.setClickListener(onItemClick); // 单项点击回调

    mProfilesList.setLayoutManager(new LinearLayoutManager(mContext));
    mProfilesList.setAdapter(mProfilesAdapter);
    onShow();
}
@Override
    void onShow() {
    IntentFilter intentFilter = new IntentFilter();
    intentFilter.addAction("GEAR_NOT_IN_PARK");
    intentFilter.addAction("GEAR_IN_PARK");
    LocalBroadcastManager.getInstance(mContext).registerReceiver( mGearReceiver, intentFilter );
    intentFilter = new IntentFilter();
    intentFilter.addAction(ACTION_REFRESH_USER_INFO); //
    mContext.registerReceiverAsUser( mProfilesChangedReceiver, UserHandle.SYSTEM, intentFilter, null, null );
    mHandler.post( mRefreshProfilesUI );
}
@Override
    void onHide() {
    mHandler.removeCallbacks( mRefreshProfilesUI );
    mContext.unregisterReceiver( mProfilesChangedReceiver );
    LocalBroadcastManager.getInstance(mContext).unregisterReceiver(mGearReceiver);
}
private final BroadcastReceiver mGearReceiver = new BroadcastReceiver() {
    @Override
    public void onReceive(Context context, Intent intent) {
        Log.d(TAG, "onReceive gear status" + intent.getAction());
        mHandler.post( mRefreshProfilesUI );
    }
};
private BroadcastReceiver mProfilesChangedReceiver = new BroadcastReceiver() {
    @Override
    public void onReceive(Context context, Intent intent) {
        Log.d(TAG, "received intent" + intent.getAction());
        mHandler.post( mRefreshProfilesUI );
    }
};
private Runnable mRefreshProfilesUI = new Runnable() {
    @Override
    public void run() {
        mHandler.removeCallbacks(mRefreshProfilesUI);
        // !!! Hack !!!  LucidSysUiService has gear controller that populates Settings.Global.
        int gear = Settings.Global.getInt( mContext.getContentResolver(),
                                           "GEAR_STATE", GearController.GEAR_PARK );
        boolean isUserStopping = mProfilesAdapter.isUserStopping();
        // 如果用户已经发动车, 或者是切换用户前的前用户, 则视为前用户 ???
        mProfilesAdapter.mDisableLogin = ( gear != GearController.GEAR_PARK || isUserStopping ); // 是否切换过用户 ?
        mProfilesAdapter.notifyDataSetChanged();

        Log.d( TAG, "mRefreshProfilesUI() : gear=" + gear + "," +
               " isUserStopping=" + isUserStopping + "," +
               " mDisableLogin=" + mProfilesAdapter.mDisableLogin );

        if ( mProfilesAdapter.mDisableLogin ) { // 如果曾经切换过用户, 那么刷新一下界面
            mHandler.postDelayed(mRefreshProfilesUI, 1500 );
        }
    }
};

ProfileListAdapter.ItemClickListener onItemClick = new ProfileListAdapter.ItemClickListener() {
    @Override
    public void onItemClick(View view, int position) {
        UserInfo mUserinf = mProfilesAdapter.getItem(position);
        if(mUserinf.id != mCarUserManagerHelper.getCurrentForegroundUserInfo().id) {
            Intent intent = new Intent();
            intent.setAction(ACTION_START_ANIMATION);
            intent.putExtra(EXTRA_USER_INFO_ID, mUserinf.id);
            intent.putExtra(EXTRA_USER_INFO_NAME, mUserinf.name);
            mContext.sendBroadcastAsUser(intent, UserHandle.SYSTEM);
            setFTEBoolean(mContext, "false"); // 这个指标，涉及某个或者某些应用是否启动
            mCarUserManagerHelper.switchToUser(mUserinf);
            // 切换用户的时候，CID 界面隐藏，播放切换动画
            LucidSystemUiUtil.sendCidNavBarCommand(mContext, CID_NAV_BAR_COMMAND_HIDE_WINDOW);
            // 当前 ICR 也隐藏，播放用户切换动画
            hideWindow();
        } else {
            Log.d(TAG, "user is same");
        }
    }
};
}
}
```

4.8 PinToDriveService.java：收到 BOOT_COMPLETED 广播后，要求当前检测到的用户密码确认身份

- 当非系统用户登录后，系统需要当前车上用户输入其账户的 5 位密码，再次确认身份无误后，方可发动系统。。。。。。
- 身份确认是在 PinToDriveModal.java，广播的接收者是服务 PinToDriveService.java
- 这里需要强调一下：当 onDestroy() 回收资源时，需要把前台运行的服务也稍微清理一下，这个有个例子，不知道是否清理干净

```
public void onDestroy() {
    Log.d( TAG, "onDestroy is called, so removing notification" );
    AsyncResources.getInstance( this ).unregisterConnections( mConnectionsListener );
    LocalBroadcastManager.getInstance(this).unregisterReceiver(mLocalBroadcastReceiver);
    unregisterReceiver(broadcastReceiver);
    stopForeground(true); // 这里会清理前台 foregroundServices(), LucidSysUiService already did this
    super.onDestroy();
}

public class PinToDriveModal extends ModalDialog {
    private void processGivenPinAndConfirm() { // 用户输完 5 位密码后的回调函数
        if (stringBuilder.length()==5){
            //Compare two pin-codes
            String hashedString = PinToDriveUtils.md5Hashing(stringBuilder.toString());
            String storedPinCode = UserProfileData.retrieveStringData(ProfileKey.SETTINGS_PIN_CODE_KEY,
                                                        PinToDriveUtils.getEncryptedLastFiveDigitsOfVin(mCont
            if (hashedString.equals(storedPinCode)){
                // Notifying BCM & ICC that user has successfully completed pin to drive process: 车上用户确认了身份
                LocalBroadcastManager.getInstance(mContext)
                    .sendBroadcast(new Intent(PinToDriveUtils.ACTION_NOTIFY_BCM_PIN_AUTH_SUCCESS));
                // 。。。
            }
        }
    }
}
public class PinToDriveService extends Service {
private Context mContext;
@Override
public void onCreate() {
    super.onCreate();
    mContext = getApplicationContext();
    enableForegroundService(); // 确保是前台服务
    AsyncResources.getInstance(this).registerConnection( mConnectionsListener );
    try {
        String jsonString = PinToDriveUtils.retrieveFailureAttemptsJsonString(mContext);
        if (jsonString == null || jsonString.length() == 0) {
            Queue<Long> queue = new PriorityQueue<>(PinToDriveUtils.TOTAL_ATTEMPTS_LIMIT);
            PinToDriveUtils.setAttemptsQueue(queue);
        } else {
            Gson gson = new Gson();
            Queue<Long> queue = gson.fromJson(jsonString, new TypeToken<PriorityQueue<Long>>() {
            }.getType());
            PinToDriveUtils.setAttemptsQueue(queue);
        }
    } catch (JsonSyntaxException e){
        Log.d(TAG,"JsonSyntaxException for Gson");
        Queue<Long> queue = new PriorityQueue<>(PinToDriveUtils.TOTAL_ATTEMPTS_LIMIT);
        PinToDriveUtils.setAttemptsQueue(queue);
    }
    IntentFilter localIntentFilter = new IntentFilter();
    localIntentFilter.addAction(PinToDriveUtils.ACTION_NOTIFY_BCM_PIN_AUTH_SUCCESS); // 监听车上当前用户确认身份
    LocalBroadcastManager.getInstance(mContext).registerReceiver(mLocalBroadcastReceiver, localIntentFilter);
    IntentFilter intentFilter = new IntentFilter();
    intentFilter.addAction(PinToDriveUtils.ACTION_RESET_ALL_TIMERS);
    registerReceiver(broadcastReceiver, intentFilter);

}
// PinToDrive helper functions
private final BroadcastReceiver broadcastReceiver = new BroadcastReceiver() {
    @Override
    public void onReceive(Context context, Intent intent) {
        Log.d(TAG,"Broadcast received for : " + intent.getAction());

        if( PinToDriveUtils.ACTION_RESET_ALL_TIMERS.equals(intent.getAction()) ){
            resetAllTimers();
        }
    }
};
private void resetAllTimers() { // 不知道这一堆稀里糊涂地在纪录什么内容。。。
    Queue<Long> queue = new PriorityQueue<>(PinToDriveUtils.TOTAL_ATTEMPTS_LIMIT); // 20 次？
    PinToDriveUtils.setAttemptsQueue(queue); // 重置回初始状态
    PinToDriveUtils.setIsPinToDriveModalShown(false); // 不再显示索要密码窗口
    PinToDriveUtils.IsTimeLimitIsReached(false);      //
    PinToDriveUtils.persistFailureAttemptsJsonString( mContext );
}
}
```

- 还记得前面 AndroidManaifest.xml 里面的申明吗？静态 STATIC BroadcastReceiver 都是这么写的吗？

```xml
<service android:name=".pinToDrive.PinToDriveService"
    android:enabled="true"
    android:launchMode="singleInstance"
    android:exported="true"/>
<receiver android:name=".pinToDrive.PinToDriveService$PinToDriveBootReceiver">
    <intent-filter>
        <action android:name="android.intent.action.BOOT_COMPLETED"/>
    </intent-filter>
</receiver>
```

- 这个服务监听 BOOT_COMPLETED 广播，当它收到 BOOT_COMPLETE 广播的时候，会执行相应的逻辑

```
public class PinToDriveService extends Service {
    // STATIC: PinToDriveService.PinToDriveBootReceiver: 设置为静态，是什么意思呢？
    public static class PinToDriveBootReceiver extends BroadcastReceiver {
        @Override
        public void onReceive(Context context, Intent intent) {
            Log.d(TAG,"onBoot completed...");
            if ( Intent.ACTION_BOOT_COMPLETED.equals( intent.getAction()) ) {
                UserManager um = (UserManager) context.getSystemService(Context.USER_SERVICE);
                if (!um.isSystemUser()) {
                    Log.d(TAG,"Starting PinToDriveService for non-system user");
                    // Starts service only for non-system user onBootCompleted
                    Intent pinToDriveIntent = new Intent(context, PinToDriveService.class);
// 当前服务是在系统收到广播后，启动自己为前台服务的
                    context.startForegroundService(pinToDriveIntent);
                }
            }
        }
    }
    private void showPinCodeModal() {
        Log.d(TAG,"Show pin code modal on CID");
```

```
            //sending a CID deploy request
            LucidSystemUiUtil.sendCidNavBarCommand(mContext, CID_NAV_BAR_COMMAND_DEPLOY_CID);
            // lucid-tray-lib 里的 CidTrayFrame.java 在监听这个广播
            sendBroadcast(new Intent(PinToDriveUtils.ACTION_ONCLICK_CID_NAV));

            Intent pinToDriveIntent = new Intent(mContext, PinToDriveActivity.class);
            pinToDriveIntent.addFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
            pinToDriveIntent.putExtra("showModalType","pin");
            PinToDriveUtils.setIsPinToDriveModalShown(true);
            LucidDisplays.startActivity(mContext, LucidDisplays.DISPLAY_CID, pinToDriveIntent);
        }
    }
```

5 LucidProfileData: 用户数据库管理应用

- 这个应用里定义了一个安卓系统层面数据库所需要的一切相关类等配置;
- 如果去看配置，就与 LucidCarService 以及 LucidUserProfile 以及 LucidProfileSetup 应用一样，是能够运行在系统用户环境下的，作为大背景来供其它用户使用的

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android" coreApp="true"
package="com.lucid.car.service"
android:sharedUserId="android.uid.system"> <!-- 开发了多个 APK 并且需要他们之间互相共享资源 -->

<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    coreApp="true"  <!-- 它是中心应用 ????? -->
    package="com.lucid.car.profiledata"
    android:sharedUserId="android.uid.system"> <!-- 开发了多个 APK 并且需要他们之间互相共享资源 -->

<manifest xmlns:android="http://schemas.android.com/apk/res/android"
          package="com.lucid.userprofile"
          android:sharedUserId="android.uid.system" <!-- 开发了多个 APK 并且需要他们之间互相共享资源 -->
          android:versionCode="23"
          android:versionName="0.3.3">

<manifest xmlns:android="http://schemas.android.com/apk/res/android"
          package="com.lucid.profilesetup"
          android:sharedUserId="android.uid.system"> <!-- 开发了多个 APK 并且需要他们之间互相共享资源 -->
<!-- 但是因为当能够检测到用户登录，车上并不总是启动这个应用，只在必要的情况下才启动，所以并不总是出现 -->
```

  - 所以，它与 LucidCarService 一样，是长存于系统用户下作为其它用户运行背景与环境配置的大后台，并被所有的用户共用。因为它本质上也不是一个数据库，这个用来存放用户数据的数据库是被这个安卓系统上所有的用户所共用的。所以，它也不会在任何非系统用户下创建，更多的可能就是进程间 bind service。再有前面提到过的，下面的日志来强加记忆一下：

```
px3lucid:/ # ps -A | grep -i com.lucid
system        2359   839 5074412  70388 SyS_epoll_wait      0 S com.lucid.car.profiledata // 这个东西看来没看懂，它是 AIDL 相
system        2488   839 5137312 104584 SyS_epoll_wait      0 S com.lucid.userprofile     // 这里定义的是 AIDL 的接口吗？
system        2699   839 5112756  55276 SyS_epoll_wait      0 S com.lucid.car.service
<<<<<<<<<<================================== 上面这两个系统层面的 Process 不曾变化，它们为多个用户所共有!!!!!!
```

- 摆一点日志在这里，看一下这几个应用 LucidUserProfile, LucidProfileData, LucidProfileSetup 以及模块 lucid-profile-lib 以及 AIDL 的 ProfileDataService.java 到底是如何相互联系起来，动态生成与动作的呢?

```
I SettingsState: ^^Iat com.android.providers.settings.SettingsProvider.onCreate(SettingsProvider.java:330)
<<<<<<<<<<============= 这里我昨时加入前面一篇的 Settings provider 的部分，
<<<<<<<<<<============= 这里不是搞不懂吗，它在一个相对更高的层面上注册了 Settings.Global ContentProvider 看这里就够了，其它可以跳
V SettingsProvider: Notifying for 0: content://settings/global/location_global_kill_switch
V SettingsProvider: Notifying for 0: content://settings/global/location_global_kill_switch

D WifiActiveModeWarden: Switching from WifiDisabledState to WifiDisabled
D VoiceInteractionManagerService: switchImplementation(0) took to complete: 0ms

I CarServiceHelper: Switching to user 10 on boot
I CAR.SERVICE: Service onCreate
```

```
I CAR.POWER: User switch disallowed while booting
D CAR.POWER: getTargetUserId(): current=10, target=UserInfo[id=10, name=Primary Driver, flags=ADMIN|INITIALIZED], isTargetP
D CAR.POWER: Not switching to 10 because it's not allowed
I AlarmClock: AlarmInitReceiver android.intent.action.LOCKED_BOOT_COMPLETED

D CarUserService: onCreate()
D VoiceInteractionManagerService: switchImplementation(0) took to complete: 4ms
I ActivityManager: Posting BOOT_COMPLETED user #0 // Headless System user 0

D CarrierProvider: onCreate 系统级别 Content Provider <<<<<<<<<<<<<<<<<========================
D CarrierIdProvider: onCreate
W ContextImpl: Calling a method in the system process without a qualified user: <<<<<<<<<<<<<<<<<========================
android.app.ContextImpl.startService:1570 android.content.ContextWrapper.startService:669
com.lucid.car.profiledata.ProfileDataApp.onCreate:20 android.app.Instrumentation.callApplicationOnCreate:1189
android.app.ActivityThread.handleBindApplication:6460
D ProfileDataService: onStartCommand(): starting service with flags= 0, startId= 1 <<<<<<<<<<<<<<<<<=======================
D ProfileDataService: onStartCommand(): initializing repository...
D ProfileDataService: onStartCommand(): starting random number generator for IM ALIVE messages...
D ProfileDataService: onStartCommand(): service has been started

D ONSAutoboot: Received android.intent.action.LOCKED_BOOT_COMPLETED
D LucidDmsService: onReceive(): android.intent.action.BOOT_COMPLETED
I SystemServiceManager: Calling switchUser u10
```

- 在任何其它应用层的应用启动起来之前，profileDataService 已经先运行起来

5.1 概述一下这个应用的主要任务：

- 这个应用走过 ProfileDataApp.java 就直接启动服务 ProfileDataService，连接数据库吗？

5.2 AndroidManifest.xml

- 这里有点儿奇怪是: 不知道是不是因为它是 AIDL IPC，这个应用它没有应用的启动 Activity ? 它没有像比如说 LucidSystemUI 应用那种应用的启动程序。就想知道，它是怎么启动起来的呢？

```
<activity
    android:name="com.lucid.car.sysui.LaunchCidActivity"
    android:icon="@mipmap/ic_launcher"
    android:label="@string/launch_cid"
    android:launchMode="singleInstance"
    android:taskAffinity=".LaunchCidActivity" >
  <intent-filter>
    <action android:name="android.intent.action.MAIN" />
    <category android:name="android.intent.category.LAUNCHER" />
  </intent-filter>
</activity>
```

- 它的 AndroidMenifest.xml 配置

```
<application
    android:name=".ProfileDataApp"
    android:label="@string/app_name"
    android:persistent="true">
  <service
      android:name=".ProfileDataService"
      android:enabled="true"
      android:exported="true"
      android:singleUser="true" />
  <service
      android:name=".SyncToCloudService"
      android:enabled="true"
      android:exported="false" />
</application>
```

5.3 SyncToCloudService.java: 处理数据云同步：云上有什么呢？数据库在云上吗？

- 这里需要理清三个服务之间的联系：ServiceNexus 提供了对其它两处服务的开启与终止 start 与 stop 的控制，以及这些进程间多线程操作的异步处理机制 ArrayBlockoingQueue 等
- SyncToCloudService 里提供了同步到数据库（这里看来还有些没有想清楚呀）的同步与否，以及若是同步，启用调用哪个同步方法（即是哪个操作 persist delete 等）各种方法参数配置
- 而这个多进程间 IPC AIDL 需要三个服务以及背后的数据库及 ArrayBlockingQueue 重入锁等协调工作来完成，感觉自己还是没有参透呀，再想一想

```
public class SyncToCloudService extends Service {
    private static final String TAG = SyncToCloudService.class.getSimpleName();
    public static final boolean ENABLE_LOG = "userdebug".equals(Build.TYPE) || "eng".equals(Build.TYPE);

    private ServiceNexus nexus = ServiceNexus.getUniqueInstance();
    // private int randomNumber;
    // private boolean rngStarted = false;
    // private long serviceUptime = 0;
    private SyncNotification stcNotification;
//*********************************************************************************************
// Sync-to-cloud methods: 应该主要就是靠这个方面把两个跨进程多线程安全的服务联接起来，背后站着一个阻塞的 ArrayBlockingQueue
// 阻塞的 ArrayBlockingQueue 通过 ReentrantLock 来实现公平生产与消费
//*********************************************************************************************
    private void monitorSyncToCloudActionQueue() {
        String mTAG = "monitorSyncToCloudActionQueue(): ";
        String stcActionMsg;
        while (true) {
            try {
// 从多线程安全的 BlockingQueue 里取出来的当前的某个 request 的 response,
                stcNotification = nexus.syncToCloudNotification.take();
// 如果这个 curr response 的参数指定云同步
                switch (stcNotification.syncToCloudAction) {
// 这里进一步指出，云同步的方法是什么呢， persiste or delete 等
                case ServiceConstants.PDS_SERVICE_STC_ACTION_PERSIST:
                    stcActionMsg = "PDS_SERVICE_STC_ACTION_PERSIST";
                    break;
                case ServiceConstants.PDS_SERVICE_STC_ACTION_DELETE:
                    stcActionMsg = "PDS_SERVICE_STC_ACTION_DELETE";
                    break;
                default: // 默认操作，不知道是作什么，数据更新吗？
                    stcActionMsg = "PDS_SERVICE_STC_ACTION_UNRECOGNIZED";
                    break;
                }
            } catch (Throwable t) { // 出错了
                ServiceUtils.logThrowableError( TAG, mTAG + "error while waiting on sync to cloud action notificatio
            }
        }
    }
    //*********************************************************************************************
    // Service lifecycle methods
    //*********************************************************************************************
    @Override public void onStart(Intent intent, int startId) {
        super.onStart(intent, startId);
    }
    @Override public void onDestroy() {
        super.onDestroy();
        stopRandomNumberGenerator();
        nexus.setSTCServiceStatus( ServiceConstants.PDS_SERVICE_STATUS_STOPPED );
    }
    @Override public int onStartCommand(Intent intent, int flags, int startId) {
        serviceUptime = 0;
        rngStarted = true;
        new Thread(new Runnable() {
                @Override public void run() {
                    startRandomNumberGenerator();
                }
            }).start();
        nexus.setSTCServiceStatus( ServiceConstants.PDS_SERVICE_STATUS_STARTED );
        monitorSyncToCloudActionQueue(); // <<<<<<<<<<<<<<<<<<<=========================================
        return START_STICKY; //
    }
    //************************************************************************************
    // Service  bind/unbind support methods
    //************************************************************************************
    @Override public IBinder onBind(Intent intent) {
        return null;
    }
    @Override public void onRebind(Intent intent) {
        super.onRebind(intent);
    }
    @Override public boolean onUnbind(Intent intent) {
        return super.onUnbind(intent);
    }
    //************************************************************************************
    // Misc support methods
    //************************************************************************************
    private void startRandomNumberGenerator() {
        while (rngStarted) {
            try {
                Thread.sleep(rngIntervalMillis);
                if (rngStarted)
                    randomNumber = new Random().nextInt(rngMax) + rngMin;
            } catch (Throwable t) {
                ServiceUtils.logThrowableError( TAG,"error while running rng thread", t );
            }
        }
    }
    private void stopRandomNumberGenerator() {
        rngStarted = false;
    }
}
```

5.4 SyncNotification.java:

```
public class SyncNotification {
    public DbProfileEntry profileEntry;
    public int syncToCloudAction;
    public boolean exceptionOccurred = false;

    public SyncNotification() { }
    public SyncNotification(DbProfileEntry profileEntry, int syncToCloudAction) {
        this.profileEntry = profileEntry;
        this.syncToCloudAction = syncToCloudAction;
    }
}
```

5.5 ServiceNexus.java: 需要再看，里面有一行代码不懂是怎么回事，整个 AIDL 跨进程的数据库操作就连不起来了!!!

- 需要再看，里面有一行代码不懂是怎么回事，整个 AIDL 跨进程的数据库操作就连不起来了!!!
- 这里没能联起来，是因为我还没有来得及看 lucid-profile-sync-lib 里的源码，把这个模块的 IPC AIDL 接口类与 LucidUserProfile 中的 SyncController.java 的实现类联系起来看，应该就能够看得懂这些个跨进程服务是如何通过 IBinder 实现跨进程操作数据库并返回其它进程 query request 的结果的了!!!
- question: 里面有一个 onfinished() 不知道是在哪里定义的，怎么联接起来的，需要找一找。因为这一块涉及到的内容比较多，再等一等，把其它源码再熟悉一下之后，再回来想这个问题。
- ArrayBlockingQueue 数据结构的基础，需要总结一下，里面还是有很多的知识点需要自己捡起来的。

```
import static com.lucid.car.profiledata.ServiceConstants.PDS_SERVICE_MAX_ABQ_SIZE_FOR_STC_REQUESTS;
// public static final int PDS_SERVICE_MAX_ABQ_SIZE_FOR_STC_REQUESTS = 30;

public class ServiceNexus {
```

```
^^Iprivate static final String TAG = ServiceNexus.class.getSimpleName();
//*******************************************************************************
// Members and methods specific to the singleton nature of this class
//*******************************************************************************
^^I// Static member holds only one instance of the RuntimeNexus class
^^Iprivate static volatile ServiceNexus c_runtimeNexusInstance;
^^I// Empty constructor prevents any other class from instantiating
^^Iprivate ServiceNexus() {^^I}
^^I// Global method that provides access the the single instance of this class
^^Ipublic static ServiceNexus getUniqueInstance() {
^^I^^I// Implement double checked locking so that we only incur the expense of synchronization once.
^^I^^I//
^^I^^I// Synchronized limits access to only one thread, other threads must await exit of first thread from the code
^^I^^I// First thread through will incur expense of synchronization. Thereafter no other thread will enter the block
^^I^^I// because the instantiation already took place.
^^I^^Iif (null == c_runtimeNexusInstance) {
^^I^^I^^Isynchronized (ServiceNexus.class) {
^^I^^I^^I^^Iif (null == c_runtimeNexusInstance)
^^I^^I^^I^^I^^Ic_runtimeNexusInstance = new ServiceNexus();
^^I^^I^^I}
^^I^^I}
^^I^^Ireturn c_runtimeNexusInstance;
^^I}
//*******************************************************************************
// Member and method declarations
//*******************************************************************************
^^I// Declarations related to service status
^^Iprivate static volatile int profileDataServiceStarted;
^^Iprivate static volatile int syncToCloudServiceStarted;

    // 用于 start 或是 stop 各个对应的 Service
^^Ipublic void setPTLServiceStatus (int status) { profileDataServiceStarted = status; };
^^Ipublic int getPTLServiceStatus () { return profileDataServiceStarted; };
^^Ipublic void setSTCServiceStatus (int status) { profileDataServiceStarted = status; };
^^Ipublic int getSTCServiceStatus () { return profileDataServiceStarted; };

^^I// 这里写得好神哦: 可是与 Android.Handler.mQueue 的 PriorityQueue 也是 blocking 阻塞与计时原理, 有什么区别呢?
    // 这里是异步, 因为分处不同的进程, 没法同步; 是使用的 Queue, 所以大致知道可能会 FIFO, 但也依赖是否设置 Priority 等是否是
    // 我先前用到那些 priorityQueue 大多不是线程安全的; 而 BlockingQueue 可以通过阻塞机制以及重入锁来实现生产者消费者等多线
    // 这里异步任务完成后的回调接口 onfinished() 是在哪里定义的呢, 如何回调工作的?连这里的注释都看不懂呀。。。。。。。。。。
^^I// Queue used by async query task in repository to pass onfinished() result back to service
^^I// - service polls on blocking queue to await result
^^I// - async task puts query result in blocking queue
^^I// - service breaks from polling on queue and return value to AIDL caller
    // 这个 blockingQueue 里面最多装 30 条 requests
^^Ipublic ArrayBlockingQueue<SyncNotification> syncToCloudNotification = new ArrayBlockingQueue<>( PDS_SERVICE_MAX_A
    //  这里仙气飘飘: 是怎么跟另外一个云同步连接起来的呢?真神了呀。。。。。。。。。
// 就这一行, 前不见古人, 后不见来者的一行, 像清歌输入法打字会飘一样, 把人看飘了。。。。。。

^^I// Map y async query task in repository to pass onfinished() result back to service
^^Iprivate static volatile Map<String, Object> profileValueQueryResultMap = new HashMap<String, Object>();
    // 这个 map 比较简单: [key, value] 分别是: string 比如说 query 字符串, value 是 query 到的结果 default 成 Object 类
    // 接下来的操作也全都是 HashMap 的基础操作: Entry 添减, get 值, 以及 remove entry
    // 只需要理解 map 的 value 存放的是 IPC AIDL 异步数据库操作的 query 结果就可以了
^^Ipublic static synchronized void profileQueryResultMapPut(String mapKey, Object result ) {
^^I^^IprofileValueQueryResultMap.put( mapKey, result );
^^I}
^^Ipublic static synchronized boolean profileQueryResultMapContains( String mapKey ) {
^^I^^Ireturn profileValueQueryResultMap.containsKey( mapKey );
^^I}
^^Ipublic static synchronized Object profileQueryResultMapGet( String mapKey ) {
^^I^^Ireturn profileValueQueryResultMap.get( mapKey );
^^I}
^^Ipublic static synchronized Object profileQueryResultMapRemove( String mapKey ) {
^^I^^Ireturn profileValueQueryResultMap.remove( mapKey );
^^I}

^^I// Declarations for random numbers intended for use by both services to see if the other is alive
^^Iprivate static volatile int profileDataServiceImAliveRandom;
^^Iprivate static volatile int syncToCloudServiceImAliveRandom;
^^Ipublic void setProfileDataServiceRandom(int status ) { profileDataServiceStarted = status; };
^^Ipublic int getProfileDataServiceRandom() { return profileDataServiceStarted; };
^^Ipublic void setSyncToCloudServiceRandom(int status ) { profileDataServiceStarted = status; };
^^Ipublic int getSyncToServiceRandom() { return profileDataServiceStarted; };
^^I// Service connection and interface declarations
^^Iprivate static volatile ServiceConnection profileDataServiceConnection;
^^Iprivate static volatile IProfileDataService iProfileDataService;
^^Ipublic void setProfileDataServiceConnection(ServiceConnection serviceConnection) {
^^I^^IprofileDataServiceConnection = serviceConnection;
^^I}
^^Ipublic ServiceConnection getProfileDataServiceConnection() {
^^I^^Ireturn profileDataServiceConnection;
^^I}
^^Ipublic void setProfileDataServiceInterface(IProfileDataService iServiceInterface ) {
^^I^^IiProfileDataService = iServiceInterface;
^^I}
^^Ipublic IProfileDataService getProfileDataServiceInterface() {
^^I^^Ireturn iProfileDataService;
^^I}
}
```

5.6 ProfileDataService.java: AIDL 跨进程服务, 前面 lucid-profile-lib 提到的 AIDL 服务 IProfileDataService AIDL 接口的实现类：数据库的创建与维护用户操作管理

- 它是一个服务，就具备了服务的生命周期，需要实现必要的生命周期函数，用于资源的妥善管理;
- 它还是一个跨进程服务，那么它就需要实现 AIDL 接口类中所定义的所有接口，用于跨进程服务 Server 与 Client 端的对接，数据交互；还记得需要操作那个 profiles 用户数据表的几个操作接口吗?
- 它还是一个跨进程服务，那么它也就需要实现，在 AIDL 跨进程过程中，服务连接过程中可能出现的错误与意外的处理
- 大致想要这么多，改天想到或是能理解得更深时，再回来补充

```
public class ProfileDataService extends Service {
    private static final String TAG = ProfileDataService.class.getSimpleName();
    public static final boolean ENABLE_LOG = "userdebug".equals(Build.TYPE) || "eng".equals(Build.TYPE);
    private ServiceNexus nexus = ServiceNexus.getUniqueInstance();
    private Intent syncToCloudServiceIntent = null;
    private ProfileRepository pdsRepository; // ProfileRepository 有一个静态 static 的所有用户共用的用户数据数据库
//*****************************************************************************************
// Service lifecycle methods
//*****************************************************************************************
    @Override public void onStart(Intent intent, int startId) { // 这是什么意思呢？
        super.onStart(intent, startId);
    }
    @Override public void onDestroy() {
        String mTAG = "onDestroy() ";
        super.onDestroy();
        stopRandomNumberGenerator();
// Coded but not using stc service for now
// stopSTCService();
        // 用来设置这个特定服务的 start 与 stop: 同步停止另一个服务 ServiceNexus
        nexus.setPTLServiceStatus( ServiceConstantsPDS_SERVICE_STATUS_STOPPED );
    }
// 当 ProfileDataApp.java 调用 startService(new Intent(this, ProfileDataService.class)); 这个方法就会被调用
    @Override public int onStartCommand(Intent intent, int flags, int startId) {
        String mTAG = "onStartCommand(): ";
        serviceUptime = 0;
        rngStarted = true;
        // 数据库是唯一的，全局共用，它只是建了一个操作数据库的类的 instance
        pdsRepository = new ProfileRepository(getApplication());
        new Thread(new Runnable() { // 只干一件事：生成随机数，异步执行，可是它为什么需要生成随机数呢？
                @Override public void run() {
                    startRandomNumberGenerator();
                }
            }).start();
// Coded but not enabled for now
// startSTCService();
```

```
        // 用来设置这个特定服务的 start 与 stop: 同步开始另一个云同步服务 ServiceNexus
        nexus.setPTLServiceStatus( ServiceConstantsPDS_SERVICE_STATUS_STARTED );
        return START_STICKY;
    }
//*******************************************************************************************
// Service bind/unbind support methods
//*******************************************************************************************
    @Override public IBinder onBind(Intent intent) {
        String mTAG = "onBind(): ";
        return iptlServiceBinder;
    }
    @Override public void onRebind(Intent intent) {
        String mTAG = "onBind(): ";
        superonRebind(intent);
    }
    @Override public boolean onUnbind(Intent intent) {
        String mTAG = "onUnbind(): ";
        return superonUnbind(intent);
    }
//*******************************************************************************************
// Misc service support methods
//*******************************************************************************************
    private void startRandomNumberGenerator() {
        while (rngStarted) {
            try {
                Threadsleep(rngIntervalMillis);
                if (rngStarted)
                    random_number = new Random()nextInt(rngMax) + rngMin;
            } catch (Throwable t) {
                ServiceUtilslogThrowableError( TAG, "error while running rng thread", t);
            }
        }
    }
    private void stopRandomNumberGenerator() {
        rngStarted = false;
    }
//*******************************************************************************************
// AIDL implementation methods
//*******************************************************************************************
    private final IProfileDataServiceStub iptlServiceBinder = new IProfileDataService.Stub() {
        @Override
        public void persistProfileData( String userId, int profileType, String profileKey, String value, boolean sync
            String mTAG = "persistProfileData(): ";
            try {
                deleteLoggingEnabled = false;
                deleteProfileData( userId, profileType, profileKey, syncToCloud );
                deleteLoggingEnabled = true;
                pdsRepositoryinsertProfileData( new DbProfileEntry( userId, profileType, profileKey, value, syncToCl
            } catch (Throwable t) {
                handleAidlExceptionError(mTAG, t);
            }
        }
        @Override
        public String retrieveProfileData( String userId, int profileType, String profileKey, String defaultValue) th
            String mTAG = "retrieveProfileData(): ";
            DbProfileEntry profileEntry;
            String retrievedData = null;
            try {
                profileEntry = pdsRepositoryretrieveProfileData(userId, IntegertoString(profileType), profileKey );
                retrievedData = profileEntrygetValue();
            } catch (Throwable t) {
                handleAidlExceptionError(mTAG, t);
            }
            return retrievedData;
        }
        @Override
        public void deleteProfileData(String userId, int profileType, String profileKey, boolean syncToCloud ) throws
            String mTAG = "deleteProfileData(): ";
            try {
                pdsRepositorydeleteProfileData( userId, IntegertoString(profileType), profileKey, deleteLoggingEnable
            } catch (Throwable t) {
                handleAidlExceptionError(mTAG, t);
            }
        }
        @Override
        public void deleteProfile(String userID, boolean syncToCloud) throws RemoteException {
            String mTAG = "deleteProfile(): ";
            try {
                pdsRepositorydeleteProfile( userID );
// Deal with syncToCloud flag here if/when implemented
            } catch (Throwable t) {
                handleAidlExceptionError(mTAG, t);
            }
        }
    };
//*****************************************************************************************
// AIDL exception handling  methods
//*****************************************************************************************
    void handleAidlExceptionError(String mTAG, Throwable t ) {
        ServiceUtilslogThrowableError( TAG, mTAG + "error while processing AIDL call", t);
    }
//*****************************************************************************************
// Sync-to-cloud service management methods
//*****************************************************************************************
    void startSyncToCloudService() {
        String mTAG = "startSyncToCloudService(): ";
        try {
            syncToCloudServiceIntent = ServiceUtilsgetServiceIntent( this, ServiceConstantsSYNC_TO_CLOUD_SERVICE_CLAS
            startService( syncToCloudServiceIntent );
        } catch( Throwable t ) {
            ServiceUtilslogThrowableError( TAG,mTAG + "error while attempting to start service", t );
        }
    }
    void stopSyncToCloudService() {
        String mTAG = "stopSTCService(): ";
        try {
            stopService( syncToCloudServiceIntent );
        } catch( Throwable t ) {
            ServiceUtilslogThrowableError( TAG, mTAG + "error while attempting to stop STC service", t );
        }
    }
}
```

5.7 IProfileDataService.java: AIDL 定义的公用接口

- 因为是 AIDL 接口，需要跨进程通信，那么 IBinder 想要 bind 到这个服务进程的其它 client 进程服务，就必须得实现这些 AIDL 定义好的跨进程接口，以便不同进程之间，ActivityManagerService 所注册过的 Services Server 进程与也向管理者注册过监听的 Client 进程之间拥有共同的数据协议，可以方便地进行数据交互
- 所以也就有，贴在上面的 ProfileDataService。java 作为 Client 的时候需要实现的 AIDL 所定义的接口方法，见上面一节。

```
interface IProfileDataService {
    void persistProfileData(String userID, int profileType, String profileKey, String profileData, boolean syncToClou
    String retrieveProfileData(String userID, int profileType, String profileKey, String defaultValue);
    void deleteProfileData(String userID, int profileType,  String profileKey, boolean syncToCloud);
    void deleteProfile(String userID, boolean syncToCloud);
}
```

5.8 ProfileRepository.java: 就也是最普通的数据库相关的操作

```
public class ProfileRepository {
    private static final String TAG = ProfileRepository.class.getSimpleName();
    // public static final boolean ENABLE_LOG = "userdebug".equals(Build.TYPE) || "eng".equals(Build.TYPE);

    static RoomDb db;
    //*****************************************************************************************
    // Db reference and initialization methods
    //*****************************************************************************************
    public ProfileRepository(Context context) {
        db = RoomDb.getDatabase(context);
    }
    //*****************************************************************************************
```

```java
    // Methods to interact with db via dao
    //*********************************************************************************************
    public synchronized boolean insertProfileData(DbProfileEntry setting ) {
        String mTAG = "insertProfileData(): ";
        boolean returnValue = true;
        try {
            db.pdsDbSettingsDao().insertProfileEntry( setting );
        } catch( Throwable t ) {
            handleDaoException( TAG, t);
            returnValue = false;
        }
        return returnValue;
    }
    public synchronized DbProfileEntry retrieveProfileData(String userId, String profileType, String profileKey) {
        String mTAG = "retrieveProfileData(): ";
        DbProfileEntry dbProfileEntry;
        DbProfileEntry defaultProfileEntry = new DbProfileEntry( userId, Integer.valueOf( profileType ), profileKey, null,
        try {
            List<DbProfileEntry> queryResults = db.pdsDbSettingsDao().retrieveProfileEntry( userId, profileType, profileKey
            if (queryResults.size() > 0) {
                dbProfileEntry = queryResults.get(0);
            } else {
                dbProfileEntry = defaultProfileEntry;
            }
        } catch( Throwable t ) {
            dbProfileEntry = defaultProfileEntry;
            handleDaoException( TAG, t);
        }
        return dbProfileEntry;
    }
    public synchronized boolean deleteProfileData(String userId, String profileType, String profileKey, boolean deleteLoggi
        String mTAG = "deleteProfileData(): ";
        boolean returnValue = true;
        try {
            db.pdsDbSettingsDao().deleteProfileEntry( userId, profileType, profileKey );
        } catch( Throwable t ) {
            handleDaoException( TAG, t);
            returnValue = false;
        }
        return returnValue;
    }
    public synchronized boolean deleteProfile(String userId ) {
        String mTAG = "deleteProfileData(): ";
        boolean returnValue = true;
        try {
            db.pdsDbSettingsDao().deleteProfile( userId );
        } catch( Throwable t ) {
            handleDaoException( TAG, t);
            returnValue = false;
        }
        return returnValue;
    }
    //*********************************************************************************************
    // Exception handling support methods
    //*********************************************************************************************
    private void handleDaoException( String TAG, Throwable t ) {
        ServiceUtils.logThrowableError( TAG, "error while running DAO async task", t);
    }
}
```

5.9 DbProfileEntryDao.java

```java
// table name: profiles
// Columns: userID, profileType, profileKey
@Dao
public interface DbProfileEntryDao {
    @Insert
    void insertProfileEntry(DbProfileEntry profileEntry );

    @Query("SELECT * FROM profiles WHERE userId = :USER_ID AND profileType = :PROFILE_TYPE AND profileKey = :PROFILE_KEY")
    List<DbProfileEntry> retrieveProfileEntry(String USER_ID, String PROFILE_TYPE, String PROFILE_KEY );

    @Query("DELETE FROM profiles WHERE userId = :USER_ID AND profileType = :PROFILE_TYPE AND profileKey = :PROFILE_KEY")
    void deleteProfileEntry(String USER_ID, String PROFILE_TYPE, String PROFILE_KEY );
    @Query("DELETE FROM profiles WHERE userId = :USER_ID")
    void deleteProfile(String USER_ID);
}
```

5.10 ProfileDataApp.java: 启动前面提到的 AIDL 服务 IProfileDataService 服务？这里调的是自己应用中的实现类？

- 想一下，前面别人注解的，同一时间就完成了，自动完成了。。。。。。

```
public class ProfileDataApp extends Application {
    private static final String TAG = ProfileDataApp.class.getSimpleName();
    @Override
        public void onCreate() {
        super.onCreate();
        try {
            startService( new Intent( this, ProfileDataService.class ) );
        } catch( Exception e ) {
            Log.e( TAG, m_TAG + "Error while attempting to start PersistToLocal service, exception inof follows:" + e
        }
    }
}
```

6 LucidUserProfile：与 LucidProfileData 和 LucidCarService 一样，所处的层级比较高

- 与 LucidProfileData /LucidProfileSetup 和 LucidCarService 一样，所处的层级比较高，处在大后台，为所有其它用户提供注册帐户，用户数据保存到数据库等大后台相关操作。前面提过，这里不重复了。
- 这个应用的内容比较多，它自己封闭了供自己使用的两个模块库，其中一个还是跨进程的 AIDL 库用于与跨进程服务建立联接。
- 这四个层级比较高一点儿的应用，应该把它们的原理都弄清楚了
- 安卓车载 DMS 是一个驾驶监测系统，安装有摄像头来描扫驾驶员的行为，好像是用户的选择：注册登录或是登出也是 DMS 的硬件状态密切相关，所以这里有些上下文的连接工作在 DMS 相关的地方。是否与调控摄像头相关呢?
- 这个应用监听 BOOT_COMPLETE 广播，收到广播信息后，开启 LucidDmsService.java 服务，用来处理同用的账户管理相关事件：登录登出，设置更改密码与 PIN 等，偏好设置，云同步偏好设置等等相关事件。涉及到的广播监听的权限管理也比较多。
- 注意这个广播接收器的优先级设置为 3000（<intent-filter android:priority="3000">），目前看来是应用层里所有应用中优先级最高的，也就是这个 LucidUserProfile 会是最先收到这个广播的应用，也会是最启动运行的应用层应用。

6.1 AndroidManifest.xml

- 这个可以参考一下前面的 LucidProfileData 应用中的这一部分，里面有比较详细的分析。

```
<permission
    android:name="com.lucid.permission.RECEIVE_USERPROFILE_INTENTS"
    android:label="lucid_userprofile_permission"
    android:protectionLevel="signature" />
<permission android:name="com.lucid.permission.START_AUTH_SERVICE"
            android:label="lucid_authentication_service_permission"
^^I         android:protectionLevel="signature"/>
<permission android:name="com.lucid.permission.START_SYNC_SERVICE"
            android:label="lucid_syncservice_permission"
            android:protectionLevel="signature"/>
<application
    android:name=".LucidUserProfileApplication"
    android:allowBackup="true"
    android:icon="@mipmap/ic_launcher"
    android:label="@string/app_name"
    android:roundIcon="@mipmap/ic_launcher_round"
    android:supportsRtl="true"
    android:persistent="true" <!-- 这种应用会顽固地运行于系统之中, 从系统一启动, 一直到系统关机 -->
  android:directBootAware="true"
  android:theme="@style/AppTheme">
  <service android:name="com.lucid.userprofile.services.LucidDmsService"
           android:exported="true"
           android:singleUser="true"/>
  <receiver android:name="com.lucid.userprofile.services.LucidDmsService$BootReceiver"
            android:exported="true">
    <intent-filter android:priority="3000">
      <action android:name="android.intent.action.BOOT_COMPLETED" />
    </intent-filter>
  </receiver>
  <service android:name=".services.TriggerScanService"
           android:directBootAware="true"
           android:enabled="true"
           android:launchMode="singleInstance"
           android:exported="true"/>
  <service android:name=".services.LucidAuthenticationService" <!-- 找不到这个 remote 服务在哪里? -->
           android:process=":remote"
           android:permission="com.lucid.permission.START_AUTH_SERVICE"
           android:exported="true">
    <intent-filter>
      <action android:name="com.lucid.userprofile.AuthService" />
^^I</intent-filter>
  </service>
  <service android:name=".services.UserDataSyncService"
           android:process=":remote"
           android:permission="com.lucid.permission.START_SYNC_SERVICE"
           android:exported="true">
    <intent-filter>
      <action android:name="com.lucid.userprofile.SyncService" />
    </intent-filter>
  </service>
</application>
```

6.2 PrimaryDriverProfileCallback.java: 这个应用定义的 public interface

```
package com.lucid.userprofile;
public interface PrimaryDriverProfileCallback {
    void profileIsLoggedInAfterBootUp();
}
```

6.3 LucidDmsService.java 服务, 用来创建新用户

6.3.1 LucidDmsService Intents Interface

- 这里定义了大量的用户帐户相关操作，这里就列举一下，以供自己查找的时候方便
- 所有的都是省略了的都是 public static final String 的，写了 private 的就是私有的，这里就不一一列举了

```
//============================================================
// LucidDmsService Intents Interface
//============================================================
ACTION_SCAN_DMS_PROFILE = "com.lucid.car.dms.ACTION_SCAN_DMS_PROFILE";
ACTION_LOGOUT_DMS_PROFILE = "com.lucid.car.dms.ACTION_LOGOUT_DMS_PROFILE";
ACTION_DELETE_DMS_PROFILE = "com.lucid.car.dms.ACTION_DELETE_DMS_PROFILE";
ACTION_DELETE_ALL_DMS_PROFILE = "com.lucid.car.dms.ACTION_DELETE_ALL_DMS_PROFILE";
ACTION_ENROLL_DMS_PROFILE = "com.lucid.car.dms.ACTION_ENROLL_DMS_PROFILE";
EXTRA_USER_PROFILE_NAME = "com.lucid.userprofile.EXTRA_USER_PROFILE_NAME";
EXTRA_AVATAR_INFO = "com.lucid.userprofile.EXTRA_AVATAR_INFO";
ACTION_REFRESH_USER_INFO = "com.lucid.userprofile.ACTION_REFRESH_USER_INFO";
private ACTION_RESET_DMS = "com.lucid.car.dms.ACTION_RESET_DMS";
```

```
    // Intent for user_starting and user_stopping
    ACTION_USER_UNLOCKED = "android.intent.action.USER_UNLOCKED";
    ACTION_USER_STOPPED = "android.intent.action.USER_STOPPED";
    EXTRA_USER_HANDLE ="android.intent.extra.user_handle";

    // Custom intents to notify Lucid Apps
    private ACTION_USER_LOGGED_IN = "com.lucid.intent.action.USER_LOGGED_IN";
    private ACTION_USER_LOGGED_OUT = "com.lucid.intent.action.USER_LOGGED_OUT";
    private EXTRA_USER_ID = "com.lucid.intent.extra.USER_ID";
    private LUCID_APPS_PERMISSION = "com.lucid.permission.RECEIVE_USERPROFILE_INTENTS";
    private EXTRA_USER_PROFILE_PHOTO_URL = "EXTRA_FTE_PROFILE_PHOTO_URL";
    private ACTION_NEW_PROFILE_CREATED = "com.lucid.userprofile.ACTION_NEW_PROFILE_CREATED";

    // Intent from Settings app to create new user profile with the given username string
    ACTION_CREATE_NEW_USER_PROFILE = "com.lucid.userprofile.ACTION_CREATE_NEW_USER_PROFILE";
    // Intent from Settings app to create new user profile with the default username
    ACTION_CREATE_NEW_DEFAULT_USER_PROFILE = "com.lucid.userprofile.ACTION_CREATE_NEW_DEFAULT_USER_PROFILE";

    // Intent from LucidCarSettings app to show UI for FTE steps that user has completed
    private ACTION_UPDATE_USER_INFO = "com.lucid.car.carsettings.ACTION_UPDATE_USER_INFO";
    // Intent from LucidCarSettings app to show dms UI screens on ICR
    private ACTION_SHOW_ENROLLMENT_UI = "com.lucid.car.dms.ACTION_SHOW_ENROLLMENT_UI";
    private ACTION_REMOVE_ENROLLMENT_UI = "com.lucid.car.dms.ACTION_REMOVE_ENROLLMENT_UI";

    // Intent from LucidCarService to login/logout to/from user profile
    private ACTION_LOGIN_TO_USER_PROFILE = "com.lucid.userprofile.ACTION_LOGIN_TO_USER_PROFILE";
    private ACTION_LOGOUT_FROM_USER_PROFILE = "com.lucid.userprofile.ACTION_LOGOUT_FROM_USER_PROFILE";
    private ACTION_START_ANIMATION = "com.lucid.user.profile.ACTION_START_ANIMATION";
    private ACTION_STOP_ANIMATION = "com.lucid.user.profile.ACTION_STOP_ANIMATION";
    private EXTRA_USER_INFO_ID = "USER_INFO_ID";
    private EXTRA_USER_INFO_NAME = "USER_INFO_NAME";

    // Intent to/from LucidCarSettings app when requesting pin reset authorization from cloud vehicle gateway for seconda
    ACTION_RESET_PIN_REQUEST = "com.lucid.car.carsettings.ACTION_RESET_PIN_REQUEST";
    ACTION_RESET_PIN_CLOUD_REQUEST_STATUS = "com.lucid.car.carsettings.ACTION_RESET_PIN_CLOUD_REQUEST_STATUS";
    ACTION_CANCEL_RESET_PIN_REQUEST = "com.lucid.car.carsettings.ACTION_CANCEL_RESET_PIN_REQUEST";
    EXTRA_RESET_PIN_REQUEST_ALLOWED = "EXTRA_RESET_PIN_REQUEST_ALLOWED";
    EXTRA_SECONDARY_PROFILE_NAME = "EXTRA_SECONDARY_PROFILE_NAME";
    private ACTION_READ_BLUETOOTH_INFO = "com.lucid.car.sysui.ACTION_READ_BLUETOOTH_INFO";

    //================================================================
    // Constants for cloud pin reset request for secondary user
    //================================================================
    VEHICLE_CLOUD_GATEWAY = "gateway.franz.qa.infotainment.pdx.atieva.com";
    VEHICLE_CLOUD_GATEWAY_PORT = "80";
    VEHICLE_TEST_LOGIN_NAME = "QA1_Info@lucid.com";
    VEHICLE_TEST_LOGIN_PASSWORD = "Info_QA1";
    VEHICLE_TEST_LOGIN_VIN = "INFO02";
```

6.3.2 LucidDmsService.mReceiver 成员变量的创建与管理

```
private void registerReceiver() {
    IntentFilter filter = new IntentFilter();
    filter.addAction( ACTION_SCAN_DMS_PROFILE );
    filter.addAction( ACTION_LOGOUT_DMS_PROFILE );
    filter.addAction( ACTION_DELETE_DMS_PROFILE );
    filter.addAction( ACTION_DELETE_ALL_DMS_PROFILE );
    filter.addAction( ACTION_ENROLL_DMS_PROFILE );
    filter.addAction( ACTION_RESET_DMS );
    filter.addAction( ACTION_USER_UNLOCKED );
    filter.addAction( ACTION_USER_STOPPED );
    filter.addAction( ACTION_SHOW_ENROLLMENT_UI );
    filter.addAction( ACTION_REMOVE_ENROLLMENT_UI );
    filter.addAction( ACTION_UPDATE_USER_INFO );
    filter.addAction( ACTION_START_ANIMATION );
    filter.addAction( ACTION_STOP_ANIMATION );
    filter.addAction( ACTION_RESET_PIN_REQUEST );
    filter.addAction( ACTION_CANCEL_RESET_PIN_REQUEST );
    filter.addAction( Intent.ACTION_LOCALE_CHANGED );
    filter.addAction( ACTION_LOGIN_TO_USER_PROFILE );
    filter.addAction( ACTION_LOGOUT_FROM_USER_PROFILE );
    getApplicationContext().registerReceiverAsUser( mReceiver, UserHandle.ALL, filter, (String)null, (Handler)null );
}
```

```
private void unregisterReceiver() {
    unregisterReceiver( mReceiver );
}
private BroadcastReceiver mReceiver = new BroadcastReceiver() {
    @Override
    public void onReceive(Context context, Intent intent) {
        // Debugging info.
        Log.d(TAG, "On received HMI action: " + intent.getAction());
        if (ACTION_SCAN_DMS_PROFILE.equals(intent.getAction())) {
            mStateMachine.actionReceived(StateMachine.ACTION.REQUEST_SCAN, null);
        } else if (ACTION_RESET_DMS.equals(intent.getAction())) {
            mStateMachine.actionReceived(StateMachine.ACTION.REQUEST_RESET, null);
        } else if (ACTION_LOGOUT_DMS_PROFILE.equals(intent.getAction())) {
            mStateMachine.actionReceived(StateMachine.ACTION.REQUEST_LOGOUT, null);
        } else if (ACTION_DELETE_DMS_PROFILE.equals(intent.getAction())) {
            mStateMachine.actionReceived(StateMachine.ACTION.REQUEST_DELETE, intent.getExtras());
        } else if ( ACTION_DELETE_ALL_DMS_PROFILE.equals( intent.getAction() ) ) {
            mStateMachine.actionReceived(StateMachine.ACTION.REQUEST_DELETE_ALL, intent.getExtras());
        } else if ( ACTION_SHOW_ENROLLMENT_UI.equals( intent.getAction() ) ) {
            FaceScanUi.getInstance().showEnrollmentScreen();
        } else if ( ACTION_ENROLL_DMS_PROFILE.equals( intent.getAction() ) ) {
            mStateMachine.actionReceived(StateMachine.ACTION.REQUEST_ENROLL, intent.getExtras());
        } else if ( ACTION_REMOVE_ENROLLMENT_UI.equals( intent.getAction() ) ) {
            FaceScanUi.getInstance().removeView();
        } else if (ACTION_USER_UNLOCKED.equals(intent.getAction())) { //  《《《===== ACTION_USER_UNLOCKED
            if(intent.hasExtra(EXTRA_USER_HANDLE)){
                int userId = intent.getIntExtra(EXTRA_USER_HANDLE,-1);
                sendStickyBroadcastAsUser( new Intent(ACTION_READ_BLUETOOTH_INFO), UserHandle.of(userId) );
                Intent userProfileIntent = new Intent(ACTION_USER_LOGGED_IN);
                userProfileIntent.putExtra(EXTRA_USER_ID, userId);
                sendBroadcastAsUser( userProfileIntent, UserHandle.of(userId), LUCID_APPS_PERMISSION  );
                Log.d(TAG,"Got a handle for user: "+ userId);
                if (ProfileLauncher.getFTEBoolean(context))
                    notifyLucidAppsForProfileCreation();
            }
        } else if (ACTION_LOGIN_TO_USER_PROFILE.equals(intent.getAction())) { //  《《《===== ACTION_LOGIN_TO_USER_PROFILE
            if(!mProfileSwitchingCIDAnimation.isProgressShown()) {
//                    mProfileSwitchingICRAnimation.showProfileCreationAnimation();
//                    mProfileSwitchingCIDAnimation.showProfileCreationAnimation();
            }
            mKeyFobUtil.switchUserForKeyFobDetection();
        } else if (ACTION_LOGOUT_FROM_USER_PROFILE.equals(intent.getAction())) { //  《《《===== ACTION_LOGOUT_FROM_USER_PROFIL
            mActivityManager.switchUser(UserProfileUtil.SYSTEM_USER);
        } else if (ACTION_USER_STOPPED.equals(intent.getAction())) {
            if (intent.hasExtra(EXTRA_USER_HANDLE)){
                int userId = intent.getIntExtra(EXTRA_USER_HANDLE,-1);
                Intent userProfileIntent = new Intent(ACTION_USER_LOGGED_OUT);
                userProfileIntent.putExtra(EXTRA_USER_ID, userId);
                sendBroadcastAsUser( userProfileIntent, UserHandle.of(userId), LUCID_APPS_PERMISSION );
                Log.d(TAG,"Got a handle for user: "+ userId);
            }
        } else if (ACTION_UPDATE_USER_INFO.equals(intent.getAction())) {
            String profileName = intent.getStringExtra(EXTRA_USER_PROFILE_NAME);
            int imageId = intent.getIntExtra(EXTRA_AVATAR_INFO, 0);
            if(!TextUtils.isEmpty(profileName)){
                mUserProfileUtil.setCurrentUserInformation(profileName, imageId);
                sendBroadcast(new Intent(ACTION_REFRESH_USER_INFO));
            }
        } else if (ACTION_START_ANIMATION.equals(intent.getAction())) {
            if(intent.hasExtra(EXTRA_USER_INFO_ID) && intent.hasExtra(EXTRA_USER_INFO_NAME)){
                int id = intent.getIntExtra(EXTRA_USER_INFO_ID, 0);
                Bitmap bitmap = mUserProfileManager.getUserIcon(id);
                String userName = intent.getStringExtra(EXTRA_USER_INFO_NAME);
                mProfileSwitchingICRAnimation.showProfileSwitchingAnimation(AppUtil.getApplicationContext().
                                                                            getString(R.string.profile_switching_animation_
                mProfileSwitchingCIDAnimation.showProfileSwitchingAnimation();
            }
        } else if (ACTION_STOP_ANIMATION.equals(intent.getAction())) {
            if (mProfileSwitchingCIDAnimation.isProgressShown()) {
                Log.d(TAG, "start remove Progress bars on ACTION_STOP_ANIMATION");
                //code to synchronize removing progressbars on CID and ICR, since WindowManage is taking to remove view on
                mHandler.postDelayed(new Runnable() {
                        @Override
                            public void run() {
                            mProfileSwitchingICRAnimation.removeAnimation();
                        }
                    }, ANIMATION_TIMEOUT_ON_PROFILE_SWITCH+1000);
                mHandler.postDelayed(new Runnable() {
                        @Override
                            public void run() {
                            mProfileSwitchingCIDAnimation.removeAnimation();
                        }
                    }, ANIMATION_TIMEOUT_ON_PROFILE_SWITCH);
            }
        } else if ( ACTION_RESET_PIN_REQUEST.equals( intent.getAction()) ) {
            /*
             * NOTE: The pin reset can't be completed properly until ISSW-4890 is completed. The reason
             * for this is that the pin reset gRPC call to cloud requires vehicleID  as an argument. Currently,
             * vehicleId, which is part of the automotive info returned with a successful gRPC login session,
             * is not persisted or otherwise accessible via an api.
             *
             * Upon completion, ISSW-4890 will persist the automotive info, which can subsequently be
             * retrieved for input to the pin reset gRPC call.
             *
             * For now, the reset pin task will use known valid values known valid values provided by the
             * cloud team.  Note that the vehicleId argument is passed as a null value. This
             * serves as a flag to the pin reset task indicating the vehicleID must be retrieved from the
             * from the cloud prior to making the pin reset request.
             *
             */
            if( intent.hasExtra(EXTRA_SECONDARY_PROFILE_NAME) ){
                String profileName = intent.getStringExtra(EXTRA_SECONDARY_PROFILE_NAME);
                String vehicleId = VehicleInfoSettings.getVehicleId(context.getContentResolver());
                mCloudResetPinTask.resetPin(context, vehicleId, profileName);
            }
        } else if ( ACTION_CANCEL_RESET_PIN_REQUEST.equals( intent.getAction()) ) {
            mCloudResetPinTask.cancel();
        }else if( Intent.ACTION_LOCALE_CHANGED.equals( intent.getAction()) ){
            //change the name of GUEST profile
            mUserProfileUtil.changeGuestProfileName();
        }
    }
};
```

6.3.3 LucidDmsService.BootReceiver

```
public static class BootReceiver extends BroadcastReceiver {
    @Override
        public void onReceive(Context context, Intent intent) {
        Log.d(TAG,"onReceive(): " + intent.getAction());
        if ( Intent.ACTION_BOOT_COMPLETED.equals( intent.getAction()) ) {
            UserManager um = (UserManager) context.getSystemService(Context.USER_SERVICE);
            if (um.isSystemUser()) { // 系统用户: 当系统启动完成之后, 启动当前 LucidDmsService.java 服务
                // Starts service only for system user on booting up case
                Intent dmsIntent = new Intent(context, LucidDmsService.class);
                context.startForegroundService(dmsIntent);
            } else {
                // FIXME : Deprecateing LucidAuthenticationService.  -paul 04/19/22
                // Intent authIntent = new Intent(context, LucidAuthenticationService.class);
                // context.startForegroundService(authIntent);
            }
        }
    }
}
```

6.3.4 LucidDmsService.java 的剩余的主要逻辑

```
public class LucidDmsService extends Service {
    private static final String TAG = "LucidDmsService";

    private UserProfileUtil mUserProfileUtil;
    private StateMachine mStateMachine;
    private UserProfileManager mUserProfileManager;
    private ActivityManager mActivityManager;
    private CarUserManagerHelper mCarUserManagerHelper;
    // private CloudResetPinTask mCloudResetPinTask;
    private ProfileData mProfileData;
    private Handler mHandler;
    private KeyFobUtil mKeyFobUtil;
//================================================================
// Life Cycle
//================================================================
    public LucidDmsService() {}
    @Override public IBinder onBind(Intent intent) {
        // TODO: Return the communication channel to the service.
        throw new UnsupportedOperationException("Not yet implemented");
    }
    @Override public int onStartCommand(Intent intent, int flags, int startId) {
        enableForegroundService();
        if (intent!=null){
            Log.d(TAG, "Started service with this intent" + intent.getAction());
            if (ACTION_CREATE_NEW_DEFAULT_USER_PROFILE.equals(intent.getAction())) { // 创建第一用户: Primary
                //Showing animation to hide profile creation process to the user
                mProfileSwitchingICRAnimation.showProfileCreationAnimation();
                mProfileSwitchingCIDAnimation.showProfileCreationAnimation();
                int index = StorageUtil.getInstance().getPrefixDriverNumber();
                Log.d(TAG,"got indexNumber: "+ index);
                String userName = AppUtil.getApplicationContext().
                    getString(R.string.default_profile_name) + index;
                mUserProfileUtil.createNewUserProfileForUserName(userName, index);
            } else if(ACTION_CREATE_NEW_USER_PROFILE.equals(intent.getAction())) { // 创建新用户: 2ndary or Guest
                String userName = intent.getStringExtra(EXTRA_USER_PROFILE_NAME);
                int avatarId = intent.getIntExtra(EXTRA_AVATAR_INFO, 50);
                mUserProfileUtil.setCurrentUserInformation(userName, avatarId);
                sendBroadcast(new Intent(ACTION_REFRESH_USER_INFO));
            }
        } else {
            Log.d(TAG,"Intent is null, can't add any action to start DMS service");
        }
        return super.onStartCommand(intent, flags, startId);
    }
    @Override public void onCreate() {
        super.onCreate();
        Log.d( TAG, "LucidDmsService starting up..." );
        mHandler = new Handler();
        registerReceiver();
        StorageUtil.getInstance().loadData();

        mProfileData = new ProfileData(getApplicationContext());
        mProfileData.bindService(new ProfileData.ServiceConnectionListener() {
                @Override public void onConnected() {
                    UserProfileData.setProfileDataInstance(mProfileData);
                }
                @Override public void onDisconnected() {
                    UserProfileData.setProfileDataInstance(null);
                }
            });
        mActivityManager = (ActivityManager) this.getSystemService(Context.ACTIVITY_SERVICE);
        mUserProfileUtil = new UserProfileUtil(this, new PrimaryProfileCallback()); // 设置回调: 定义贴在上面, 两个接口应该是
        mStateMachine = new StateMachine(null);
        mUserProfileManager = new UserProfileManager(this);
        mCarUserManagerHelper = new CarUserManagerHelper(this);
        mCloudResetPinTask = new CloudResetPinTask();
        mKeyFobUtil = new KeyFobUtil( this, mCarUserManagerHelper, mUserProfileUtil );
        mHandler= new Handler();
    }
    @Override public void onDestroy() {
        Log.d( TAG, "LucidDmsService shutting down..." );
        if (mProfileData.isServiceBound())
            mProfileData.unbindService();
        mStateMachine.cleanUp();
        unregisterReceiver();
        mKeyFobUtil.cleanUp();
        super.onDestroy();
    }
//================================================================
// Foreground Service
//================================================================
    private static final String CHANNEL_ID = "DMSForegroundService";
    private static final int NOTIFICATION_ID = 2; // 1 is being used by LucidAssistantService, so kept 2
    private void enableForegroundService() {
        createNotificationChannel();
        Notification notification = new Notification.Builder(this,CHANNEL_ID).setContentTitle("DMSService is available")
```

```
            .setSmallIcon(R.drawable.ic_launcher_foreground).build();
        startForeground(NOTIFICATION_ID,notification);
    }
    private void createNotificationChannel() {
        NotificationChannel serviceChannel = new NotificationChannel(
            CHANNEL_ID,
            "Notification Service",
            NotificationManager.IMPORTANCE_DEFAULT
            );
        NotificationManager manager = getSystemService(NotificationManager.class);
        if (manager != null) {
            manager.createNotificationChannel(serviceChannel);
        }
    }
//=================================================================================
// Send broadcast to HVAC app & LucidCarSettings app to notify for profile creation
//=================================================================================
    private void notifyLucidAppsForProfileCreation(){
        PackageManager packageManager = getPackageManager();
        Intent intentToFind = new Intent(ACTION_NEW_PROFILE_CREATED);
        List<ResolveInfo> matches = packageManager.queryBroadcastReceivers(intentToFind, 0);
        for (ResolveInfo resolveInfo : matches) {
            Intent intent = new Intent(new Intent(ACTION_NEW_PROFILE_CREATED));
            ComponentName componentName=
                new ComponentName(resolveInfo.activityInfo.applicationInfo.packageName,
                                  resolveInfo.activityInfo.name);
            intent.setComponent(componentName);
            Log.d(TAG, componentName.toString());
            sendBroadcastAsUser( intent, UserHandle.CURRENT );
        }
    }
//=================================================================================
// Callback for triggering DMS scan after boot-up and primary profile is logged in
//=================================================================================
    private class PrimaryProfileCallback implements PrimaryDriverProfileCallback { // 定义在上面部分
        @Override
            public void profileIsLoggedInAfterBootUp() { // 只有这一个 API
            // start the trigger dms scan service
            Log.d(TAG,"profileIsLoggedInAfterBootUp(): Primary Driver is logged in!!");
            Intent dmsServiceIntent = new Intent(getApplicationContext(), TriggerScanService.class);
            getApplicationContext().startForegroundServiceAsUser(dmsServiceIntent, UserHandle.SYSTEM);
        }
    }
}
```

6.4 com.lucid.synclib.ISyncCallBack.aidl: 是跨进程模块的同步接口类

```
// ISyncCallBack.aidl
package com.lucid.synclib;
// Declare any non-default types here with import statements
interface ISyncCallBack {
    void onNewDataAvailable(String data, String action) ;
    String onPullFullData();
}
```

- 不知道你先前注意到没有，系统用户下有一个 LucidUserProfile 的进程，非系统用户下也有个同样名字的进程，这是为什么呢？一个在系统用户下充当 Server 服务的，在非系统用户下的是 Client，可以与 Server 进行交互吗？而这样一个应用也确实定义了 IPC AIDL 的接口类，那么非系统用户下的服务充当 Client 与系统用户下的 Servier 进行交互，有什么不可以的呢？

6.5 com.lucid.synclib.ISyncService.aidl: 是跨进程模块的同步接口类

```
// ISyncService.aidl
package com.lucid.synclib;
import com.lucid.synclib.ISyncCallBack;

// Declare any non-default types here with import statements
interface ISyncService {
    // Applications can use this API to send the data and save it to the cloud.
    // Data needs to be sent in the form of a JSON payload.
```

```java
    void commitData(String data, String action);
    // Applications can check for any new updates from the cloud using this API call.
    void checkForUpdates();
    void registerCallBack(ISyncCallBack cb);
    void unregisterCallBack();
}
```

6.6 com.lucid.synclib.SyncController.java：调控与应用层各监听相关数据变化的应用建立回调注册与解绑，调控的所站立的层面相对比较高

```java
public class SyncController {
    private ISyncService mSyncService;
    private SyncServiceConnection mSyncServiceConnection = new SyncServiceConnection();
    private Context mContext;
    private SyncListener mSyncListener;

    class SyncServiceConnection implements ServiceConnection {
        @Override public void onServiceConnected(ComponentName name, IBinder iBinder) {
            Log.d(TAG, "service connected");
            mSyncService = ISyncService.Stub.asInterface(iBinder);
            try {
                mSyncService.registerCallBack(mCallback);
            } catch (RemoteException e) {
                e.printStackTrace();
            }
            mSyncListener.onConnected();
        }
        @Override public void onServiceDisconnected(ComponentName name) {
            mSyncListener.onDisconnected();
            Log.d(TAG, "service  disconnected");
            mSyncService = null;
        }
    }
// 因为这个服务会是 Server 中的 IBinder 实现类，所以他需要继承自 IBinder.Stub() 类
    private ISyncCallBack.Stub mCallback = new ISyncCallBack.Stub() {
        @Override
        public void onNewDataAvailable(String data, String action) throws RemoteException {
            Action enumAction = DataWrapper.getEnumAction(action);
            if (enumAction != null && mSyncListener != null)
                mSyncListener.onNewDataAvailable(data, enumAction);
        }
        @Override public String onPullFullData() throws RemoteException {
            if (mSyncListener != null)
                return mSyncListener.onPullFullData();
            else return null;
        }
    };
    /**
     * Call from applications to initialize the SyncService
     * @param context      Application context
     * @param syncListener SyncListener is SyncListener Interface to receive call backs
     * @return returns true if bindService() is successful
     */
    public boolean initService(Context context, SyncListener syncListener) {
        mContext = context;
        mSyncListener = syncListener;
        Intent intent = new Intent("com.lucid.userprofile.SyncService");
        intent.setPackage("com.lucid.userprofile");
        boolean ret = mContext.bindService(intent, mSyncServiceConnection, Context.BIND_AUTO_CREATE);
        if (!ret) {
            Log.e(TAG, "bind service failed");
            return false;
        } else return true;
    }
    /**
     * Call from applications to stop the service
     */
    private void releaseService() {
        try {
            mSyncService.unregisterCallBack();
        } catch (RemoteException e) {
            e.printStackTrace();
        }
        mContext.unbindService(mSyncServiceConnection);
        mSyncServiceConnection = null;
    }
    /**
     * Applications can call this method to save their data
     *
     * @param data   data JSON String with key-value pairs
     * @param action Action can be:
     *               <pre>
     UPDATE : Update will check if the key exists then it will replace otherwise
     adds the new field (can handle only 2 levels && items in it must have a id
     field to delete the data for that key)
     ARRAY_DELETE : Used to delete items for a array inside data (items in it must
     have a id field to delete the data for that key)
     For example {"id" : "unique string"}
     FULL: Can be used to replace full data
     CLEAR : full data under the application will be cleared
     </pre>
    */
    public void commitData(String data, Action action) {
        try {
            mSyncService.commitData(data, action.toString());
        } catch (RemoteException e) {
            e.printStackTrace();
        }
    }
    /**
     * Applications can call this optional api if they want to
     * check for any new updates. Even if this is not called apps can receive new data from callback
     */
    public void checkForUpdates() {
        try {
            mSyncService.checkForUpdates();
        } catch (RemoteException e) {
            e.printStackTrace();
        }
    }
    /**
     * Interface for applications to receive callbacks
     */
    public static class SyncListener {
        /**
         * callback for applications to receive new data
         * @param data   json data in the form of key-value pairs
         * @param action same as action in commitData() method
         */
        public void onNewDataAvailable(String data, Action action) {  }
        /**
         * callback for applications to know that service is connected
         */
        public void onConnected() {  }
        /**
         * callback for applications to know that service is disconnected
         */
        public void onDisconnected() {  }
        /**
         * optional callback for applications to pull whole data from cloud
         * when it connects to lucid id
         *
         * @return returns the full json payload in the form of String
         */
        public String onPullFullData() {
            return null;
        }
    }
    public enum Action {
        UPDATE("update"),
        ARRAY_DELETE("array_delete"),
        FULL("full"),
        CLEAR("clear");
        private String m_name;
        Action(String name) {
            m_name = name;
        }
        @Override public String toString() {
          return this.m_name;
        }
      }
    }
```

6.7 UserDataSyncService.java: 云同步，各种数据同步

- 主要负责用户数据 mProfileData 与用户的 home 和 work 地址数据的管理与同步

```
// 这个也与用户的工作或是 home 的导航数据也联系在一起，就不只是 ProfileData 的云同步的内容
public class UserDataSyncService extends Service {

  private Handler mUpLinkHandler, mDownLinkHandler;
  private HashMap<String, String> mNodeHashMap = new HashMap<>();
// [key, value]: 这里的 value 是 com.lucid.synclib, 也就是此 LucidUserProfile 自已包装的 IPC AIDL lib 所定义的 ISyncCal
  private HashMap<String, ISyncCallBack> mCallBackMap = new HashMap<>();
  private HandlerThread mUpLinkHandlerThread, mDownLinKHandlerThread;

  private ProfileData mProfileData;
  private boolean isConnected;
  private NavigationUtil navUtil;

  @Nullable @Override public IBinder onBind(Intent intent) {
    return mBinder;
  }
  @Override public void onCreate() {
    loadNodeData();
    mUpLinkHandlerThread = new HandlerThread("UpLinkHandlerThread");
    mDownLinKHandlerThread = new HandlerThread("DownLinkHandlerThread");
    mUpLinkHandlerThread.start();
    mDownLinKHandlerThread.start();
    // isDebuggable =  true;//( 0 != ( getApplicationInfo().flags & ApplicationInfo.FLAG_DEBUGGABLE ) );
    mUpLinkHandler = new Handler(mUpLinkHandlerThread.getLooper(), mUpLinkHandlerCallBack);
    mDownLinkHandler = new Handler(mDownLinKHandlerThread.getLooper(), mDownLinkHandlerCallBack);
// 这里说，如果接到有新数据的广播，那么就接收这些广播，并进行接下来的同步操作
    IntentFilter intentFilter = new IntentFilter();
    if (isDebuggable)
      intentFilter.addAction(ACTION_GET_NEW_DATA);
    intentFilter.addAction(NavigationUtil.NEW_HOME_ADDRESS_AVAILABLE);
    intentFilter.addAction(NavigationUtil.NEW_WORK_ADDRESS_AVAILABLE);
    registerReceiver(mReceiver, intentFilter);

    // bind PTL service early to make sure it is ready before retrieve and save dat from/to it.
    mProfileData = new ProfileData(getApplicationContext());
    // 为什么这个东西也需要与 ProfileData 联系起来呢？
    navUtil = new NavigationUtil(this, mProfileData);
// mProfileData 总是需要与远程的 IProfileData 的 Server 进行联系，以进行数据交互的
    mProfileData.bindService(new ProfileData.ServiceConnectionListener() {
        @Override public void onConnected() {
          isConnected = true;
        }
        @Override public void onDisconnected() {
          isConnected = false;
        }
      });
  }
  BroadcastReceiver mReceiver = new BroadcastReceiver() {
    @Override
    public void onReceive(Context context, Intent intent) {
      String action = intent.getAction();
      if (ACTION_GET_NEW_DATA.equals(intent.getAction())) {
        // 新数据是写在本地的一个什么文件的？需要从这里读取配置内容数据内容，再进一步地发消息同步
        String items = FileUtil.getOptionalConfigFromSDCard(intent.getStringExtra(EXTRA_JSON_FILE_NAME));
        DataWrapper.DataPackage  downLinkData;
        try {
          ArrayList<DataWrapper.DataPackage> dataPackageArrayList = DataWrapper.deconstructPayload(items);
          Log.d(TAG, "dataPackageArrayList" + dataPackageArrayList);
          if (dataPackageArrayList != null) {
            for(int i=0; i<dataPackageArrayList.size(); i++) {
              downLinkData = dataPackageArrayList.get(i);
              Message msg = mDownLinkHandler.obtainMessage();
              msg.obj = downLinkData;
              mDownLinkHandler.sendMessage(msg);
```

```
                        }
                    }
                } catch (JSONException e) {
                    e.printStackTrace();
                }
// 添加新的 home 或是 work  地址
            } else if (NavigationUtil.NEW_HOME_ADDRESS_AVAILABLE.equals(action)
                       || NavigationUtil.NEW_WORK_ADDRESS_AVAILABLE.equals(action)) {
                // only process the change intent which is not from Navigation
                if (!intent.hasExtra(NavigationUtil.FROM_NAVIGATION)) {
                    boolean isHome = NavigationUtil.NEW_HOME_ADDRESS_AVAILABLE.equals(action);
                    String data = navUtil.constructHomeOrWorkDataPayload(isHome);
                    Log.d(TAG, "sync to nav payload:" + data);
                    if (data != null) {
                        DataWrapper.DataPackage downLinkData = new DataWrapper.DataPackage();
                        downLinkData.action = SyncController.Action.UPDATE.toString(); // 这些是通过消息机制发送出去的
                        downLinkData.nodeName = NAV_NODE;
                        downLinkData.data = data;
                        Message msg = mDownLinkHandler.obtainMessage();
                        msg.obj = downLinkData;
                        mDownLinkHandler.sendMessage(msg);
                    }
                }
            }
        }
    };
    Handler.Callback mDownLinkHandlerCallBack = new Handler.Callback() {
        @Override
        public boolean handleMessage(Message message) {
            DataWrapper.DataPackage downLinkDataPackage = (DataWrapper.DataPackage) message.obj;
            Log.d(TAG, "downLinkDataPackage" + downLinkDataPackage);
            if (downLinkDataPackage != null) {
                String data = downLinkDataPackage.data;
                String action = downLinkDataPackage.action;
                String nodeName = downLinkDataPackage.nodeName;
                try {
                    Objects.requireNonNull(mCallBackMap.get(nodeName)).onNewDataAvailable(data,action);
                } catch (RemoteException e) {
                    mCallBackMap.remove(downLinkDataPackage.nodeName);
                    e.printStackTrace();
                }
            }
            return true;
        }
    };
    Handler.Callback mUpLinkHandlerCallBack = new Handler.Callback() {
        @Override // 主要处理与导航相关的逻辑
        public boolean handleMessage(Message message) {
            DataWrapper.DataPackage upLinkDataPackage = (DataWrapper.DataPackage) message.obj;
            String data = upLinkDataPackage.data;
            String action = upLinkDataPackage.action;
            String nodeName = upLinkDataPackage.nodeName;
            String jsonPayload = null;
            Log.d(TAG, "NODE_NAME:" + nodeName);
            //process navigation data
            if (NAV_NODE.equals(nodeName) && isConnected)
                navUtil.processNavigationData(action, data);
            try {
                jsonPayload = DataWrapper.constructPayload(data, nodeName, action);
            } catch (JSONException e) {
                e.printStackTrace();
            }
            if (isDebuggable)
                FileUtil.writeJsonToFile("ExampleJsonData.json", jsonPayload);
            Log.d(TAG, "jsonPayload: " + jsonPayload);
            //send the payload to the cloud using api
            return true;
        }
    };
// 当前 LucidUserProfile Instance 作为 Client 想要与 Servier 进行数据交换, 那么它就需要实现 AIDL 接口类中申明的所有的方法
    private final ISyncService.Stub mBinder = new ISyncService.Stub() {
        @Override
        public void commitData(String data, String action) {
            String pkgName = getPackageManager().getNameForUid(Binder.getCallingUid());
            DataWrapper.DataPackage upLinkData = new DataWrapper.DataPackage();
```

```
                Log.d(TAG, "packageName:" + pkgName);
                Message msg = mUpLinkHandler.obtainMessage();
                upLinkData.nodeName = getNodeName(pkgName);
                upLinkData.data = data;
                upLinkData.action = action;
                msg.obj = upLinkData;
                mUpLinkHandler.sendMessage(msg);
            }
            @Override public void checkForUpdates() throws RemoteException {
                //check for the updates in the cloud if there are any
                //if there are any new updates
                //data is from the cloud
            }
    // 这里传进来的 ISyncCallBack 接口，是一个跨进程的 ISyncCallBack。Stub IBinder 的实现，是作为客户端使用的？
    // 就是说，等跨进程数据库操作的 request process 完成了，请你通知跨进程的 ISyncCallBack。Stub IBinder 的实现服务，它会想要拿
            @Override public void registerCallBack(ISyncCallBack cb) throws RemoteException {
                Log.d(TAG, "register callback");
                String pkgName = getPackageManager().getNameForUid(Binder.getCallingUid());
                mCallBackMap.put(getNodeName(pkgName), cb);
                Log.d(TAG, "callback:" + mCallBackMap.toString());
            }
            // 跨进程服务的资源管理
            @Override public void unregisterCallBack() throws RemoteException {
                Log.d(TAG, "unregister callback");
                String pkgName = getPackageManager().getNameForUid(Binder.getCallingUid());
                mCallBackMap.remove(getNodeName(pkgName));
                Log.d(TAG, "callback:" + mCallBackMap.toString());
            }
        };
    // 这里所放置的内容，与 Lucid Air OS 启动的几个前台应用有什么关系呢？
        private void loadNodeData() {
            mNodeHashMap.put("com.lucid.home", "LucidHomeApp");
            mNodeHashMap.put("com.lucid.car.media", "LucidMediaApp");
            mNodeHashMap.put("com.lucid.assistant", "LucidAssistant");
            mNodeHashMap.put("com.lucid.userprofile", "LucidUserProfile");
            mNodeHashMap.put("com.lucid.car.sysui", "LucidSystemUI");
            mNodeHashMap.put("com.lucid.car.dialer", "LucidDialerApp");
            mNodeHashMap.put("com.lucid.car.carcontrols", "LucidCarControls");
            mNodeHashMap.put("com.lucid.car.hvac", "LucidHvacApp");
            mNodeHashMap.put("com.here.hnod.feynman", "LucidNavigationApp");
            mNodeHashMap.put("com.lucid.syncdemoapp", "SyncDemoApp");
            mNodeHashMap.put("com.lucid.examplesync", "ExampleSyncApp");
        }
        private String getNodeName(String pkgName) {
            if (mNodeHashMap.containsKey(pkgName)) {
                return mNodeHashMap.get(pkgName);
            } else {
                return pkgName;
            }
        }
        @Override public void onDestroy() {
            unregisterReceiver(mReceiver);
            if (mProfileData != null) // 这个永远是自动与远程进程 bind 在一起的
                mProfileData.unbindService();
            mUpLinkHandlerThread.quit();
            mDownLinKHandlerThread.quit();
            mUpLinkHandler.removeCallbacksAndMessages(null);
            mDownLinkHandler.removeCallbacksAndMessages(null);
        }
    }
```

6.8 UserProfileUtil.java

```
public class UserProfileUtil {

    private Context mContext;
    private UserManager mUserManager;
    private UserProfileManager mUserProfileManager;

    private static final String TAG = "UserProfileUtil";
    private static final int DEFAULT_PREFIX_INDEX = 2;
    private static final int INVALID_USER_PROFILE_ID = -1;
    public static final int SYSTEM_USER = 0;
    private PrimaryDriverProfileCallback mProfileCallback;
```

```java
public UserProfileUtil(Context context, PrimaryDriverProfileCallback profileCallback){
    mContext = context;
    mUserProfileManager = new UserProfileManager(mContext);
    mUserManager = (UserManager) mContext.getSystemService(Context.USER_SERVICE);
    mProfileCallback = profileCallback; // public private interface 通用
}
public void createNewUserProfileForUserName(String userName, int index){
    UserInfo info = mUserProfileManager.createNewUserWithNoRestrictions(userName, ProfileAccountType.SECONDARY);
    // To make default name dynamic, prefix is being generated with next int value
    // ...
}
public void setCurrentUserProfilePicture(Bitmap drawable){
    UserInfo info = mUserProfileManager.getCurrentForegroundUserInfo();
    mUserManager.setUserIcon(info.id, drawable);
}
public void setCurrentUserInformation(String userName, int avatarId){
    UserInfo info = mUserProfileManager.getCurrentForegroundUserInfo();
    mUserManager.setUserName(info.id,userName);
    mUserManager.setUserIcon(info.id,getBitmapFromAssets(avatarId));
}
/** This function manages the Guest Profile & Primary Driver creation or user switch to last used profile
 * It will check if there is any existing default Guest Profile & Primary Driver profile (admin)
 * if no, it will create one Guest Profile, Primary Driver Profile(admin) and switch to Primary Driver Profile
 * if yes, it will switch to Primary Driver Profile or last used profile
 **/
public void startDefaultPrimaryUserSession(){
    // Setting global value as false for not triggering FTE flow
    ProfileLauncher.setFTEBoolean(mContext,"false");

    String guestProfileName = mContext.getString(R.string.guest_profile_name);
    String primaryDriverName = mContext.getString(R.string.primary_profile_name);
    int primaryProfileId = getUserIdForGivenUserProfileType(ProfileAccountType.PRIMARY);
    if (primaryProfileId == INVALID_USER_PROFILE_ID) {
        UserInfo primaryUserInfo = mUserProfileManager.createNewUserWithNoRestrictions(primaryDriverName, ProfileAccoun
        mUserManager.setUserName(primaryUserInfo.id,primaryDriverName);
        mUserManager.setUserIcon(primaryUserInfo.id,getBitmapFromAssets(0));
        StorageUtil.getInstance().setPrimaryProfileUserId(primaryUserInfo.id);
        primaryProfileId = primaryUserInfo.id;
    }
    if (getUserIdForGivenUserProfileType(ProfileAccountType.GUEST) == INVALID_USER_PROFILE_ID) {
        //Setting default prefix number 2 - e.g. Driver2
        StorageUtil.getInstance().setPrefixDriverNumber(DEFAULT_PREFIX_INDEX);
        mUserProfileManager.createNewUserWithNoRestrictions(guestProfileName, ProfileAccountType.GUEST);
    }
    Log.d(TAG, "Switching to userId: " + primaryProfileId);
    if (primaryProfileId!=INVALID_USER_PROFILE_ID)
        switchUser(primaryProfileId, mProfileCallback);
}

public void switchUser(int androidId, PrimaryDriverProfileCallback callback){
    if (mUserProfileManager.switchToUserId(androidId)) {
        Log.d(TAG,"Android system is logged in with "+ androidId + " id");
        if (androidId == StorageUtil.getInstance().getPrimaryProfileUserId() ) {
            // Only assigning VIN number to Primary Profile when there is no pin code assigned
            if( UserProfileData.retrieveStringData(ProfileKey.SETTINGS_PIN_CODE_KEY, "").equals("") ){
                //Only assigning VIN number to Primary Profile
                setDefaultPinAsLastFiveDigitsOfVin();
            }
            if ( mProfileCallback!=null )
                callback.profileIsLoggedInAfterBootUp(); // 调用回调
        }
    }else
        Log.d(TAG, "User didn't switched to userId: " + androidId);
}
public void logoutFromUserProfile(){
    if(mUserProfileManager.switchToUserId(SYSTEM_USER)){
        Log.d(TAG,"Android system is logged out from the user profile");
    }else
        Log.d(TAG,"Android system isn't logged out from the user profile");
}
private Bitmap getBitmapFromAssets(int index) {    }
private int getUserIdForGivenUserProfileType(String type) {    }
public void changeGuestProfileName() {    }
private void setDefaultPinAsLastFiveDigitsOfVin() {    }
}
```

6.9 TriggerScanService.java: 启动了登录之后用户的人脸识别？成功之后终止此前台服务

```
public class TriggerScanService extends Service {
    private static final String TAG = TriggerScanService.class.getName();

    private StateMachine mStateMachine;
    private static final int TRIGGER_SCAN_TIMEOUT = 12000;

    @Override
        public void onCreate() {
        Log.d(TAG, "onCreate()");
        super.onCreate();
        mStateMachine = new StateMachine(new ScanCallback()); // 监听扫描的结果
        // Entry point to whole flow
        AsyncResources.getInstance( getBaseContext() )
            .registerConnection( mConnectionListener );
    }
    @Override
        public int onStartCommand(Intent intent, int flags, int startId) {
        Log.d(TAG,"onStartCommand()");
        enableForegroundService(); // 前台服务
        mHandler = new Handler();
        mHandler.postDelayed(new Runnable() {
                @Override
                    public void run() {
                    mIsTimeOut = true;
                    tryToTriggerFaceScan(); // 用户已经登录完成了，还再启动面部扫描人脸识别？
                }
            }, TRIGGER_SCAN_TIMEOUT);
        return super.onStartCommand(intent, flags, startId);
    }
    @Override
        public void onDestroy() {
        Log.d(TAG,"onDestroy()");
        AsyncResources.getInstance( getBaseContext() )
            .unregisterConnections( mConnectionListener );
        stopForeground(true);
        super.onDestroy();
    }
    //==================================================================
    // States management.
    //==================================================================
    private static final int DOOR_STATE_CLOSED = 2;
    private boolean mIsTimeOut, mIsScanStarted;
    private Handler mHandler;
    private DmsController.DmsListener mDmsListener = new DmsController.DmsListener(){
        @Override
        public void onUpdateHeadRecognitionStateChangedW(boolean dmuHeadRecognitionStatus) {
            AsyncResources asyncRes = AsyncResources.getInstance( getBaseContext() );
            if ( !asyncRes.isReadyAll() )
                return;
            if ( dmuHeadRecognitionStatus ){
                Log.d(TAG,"Head is recognized: try scan !!");
                tryToTriggerFaceScan();
            }
        }
    };
    private GearController.GearListener mGearListener = new GearController.GearListener(){
        @Override
        public void onUpdateGearStateChangedW(Integer state) {
            AsyncResources asyncRes = AsyncResources.getInstance( getBaseContext() );
            if ( !asyncRes.isReadyAll() )
                return;
            if ( state == GearController.GEAR_PARK ) {
                Log.d(TAG,"Gear is in park: try scan !!");
                tryToTriggerFaceScan();
            }
        }
    };
    private CarController.CarControlsListener mCarControlsListener = new CarController.CarControlsListener(){
        @Override
        public void onDriverDoorStateChangeW(int value) {
            AsyncResources asyncRes = AsyncResources.getInstance( getBaseContext() );
            if ( !asyncRes.isReadyAll() )
                return;
            if ( value == DOOR_STATE_CLOSED ){
                Log.d(TAG,"Door is closed: try scan !!");
                tryToTriggerFaceScan();
            }
        }
    };
    AsyncResources.ConnectionsListener mConnectionListener = new AsyncResources.ConnectionsListener(
        mCarControlsListener,
        mGearListener,
        mDmsListener) {
        @Override
        public void onConnectedAll() {
            tryToTriggerFaceScan();
        }
        @Override
        public void onDisconnectedAny() { // Do nothing.
        }
    };
    //================================================================
    // Try Face scan
    //================================================================
    private void tryToTriggerFaceScan(){
        Log.d(TAG,"tryToTriggerFaceScan()");
        if ( mIsTimeOut ){
            Log.d(TAG, "Due to timeout killing the service");
            stopSelf();
            return;
        }
        if ( mIsScanStarted ){
            Log.d(TAG, "Scan was started already");
            return;
        }
        // Have we satisfied all pre-requisites?
        AsyncResources asyncRes = AsyncResources.getInstance( getBaseContext() );
        if ( !asyncRes.isReadyAll() )
            return;
        boolean isDriverDoorOpen = ( asyncRes.getCarController().getDriverDoorState() != DOOR_STATE_CLOSED );
        Log.d(TAG, "DriverDoorOpen: " + isDriverDoorOpen);
        boolean isGearInPark = ( asyncRes.getGearController().getGearStatus() == GearController.GEAR_PARK );
        Log.d(TAG, "GearInPark: " + isGearInPark);
        boolean isHeadRecognized = ( asyncRes.getDmsController().isHeadRecognized() );
        Log.d(TAG,"HeadRecognized: " + isHeadRecognized);
        if ( !isDriverDoorOpen && isGearInPark && isHeadRecognized ){
            Log.d(TAG, " Got all 3 signals and starting the scan");
            startFaceScan();
        }
    }
    private void startFaceScan(){
        Log.d(TAG,"startFaceScan()");
        mIsTimeOut = false;
        mIsScanStarted = true;
        mHandler.removeCallbacksAndMessages(null);
        mStateMachine.actionReceived(StateMachine.ACTION.REQUEST_SCAN, null); // 触发状态机状态的改变
    }
    //================================================================
    // Stop Face scan
    //================================================================
    private void stopFaceScan(){
        Log.d(TAG,"stopFaceScan()");
        //send request through StateMachine to make it IDLE
        mStateMachine.actionReceived(StateMachine.ACTION.REQUEST_RESET, null); // 触发状态机状态的改变
    }
    private class ScanCallback implements ScanResultCallback { // 和前一个例子一样，也有公用接口
        @Override
            public void scanResultIsAvailable(boolean value) {
            Log.d(TAG,"Got result for scan.. destroying the service");
            stopSelf(); // 终止前台服务？
        }
    }
}
```

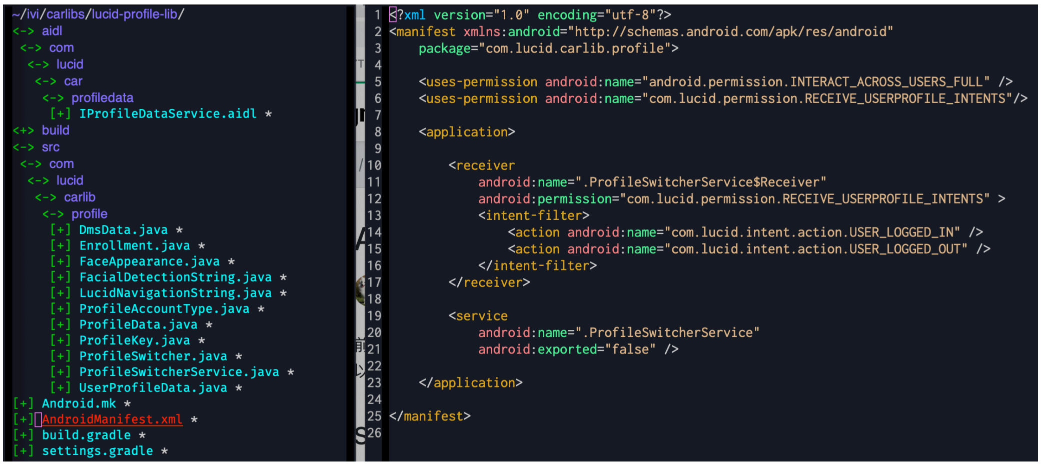
6.10 UserProfileManager.java: 监听用户切换系统发送的广播，并作相应的监听回调工作

- 类里面还有很多 setter/getter 用来对用户切换等进行管理，跳过

```
public class UserProfileManager {
    private static final String TAG = "UserProfileManager";

    public interface OnUsersUpdateListener {
        void onUsersUpdate();
    }

    private final UserManager mUserManager;
    private final ActivityManager mActivityManager;
    private ArrayList<UserProfileManager.OnUsersUpdateListener> mUpdateListeners = new ArrayList();

    public UserProfileManager(Context context) {
        mContext = context;
        mUserManager = (UserManager) mContext.getSystemService(Context.USER_SERVICE);
        mActivityManager = (ActivityManager) mContext.getSystemService(Context.ACTIVITY_SERVICE);
    }

    private final BroadcastReceiver mUserChangeReceiver = new BroadcastReceiver() {
        public void onReceive(Context context, Intent intent) {
            ArrayList copyOfUpdateListeners;
            synchronized(UserProfileManager.this.mUpdateListeners) {
                copyOfUpdateListeners = new ArrayList(UserProfileManager.this.mUpdateListeners);
            }
            Iterator var4 = copyOfUpdateListeners.iterator();
            while(var4.hasNext()) {
                UserProfileManager.OnUsersUpdateListener listener = (UserProfileManager.OnUsersUpdateListener)var4.ne
                listener.onUsersUpdate();
            }
        }
    };
    private void registerReceiver() {
        IntentFilter filter = new IntentFilter();
        filter.addAction("android.intent.action.USER_REMOVED");
        filter.addAction("android.intent.action.USER_ADDED");
        filter.addAction("android.intent.action.USER_INFO_CHANGED");
        filter.addAction("android.intent.action.USER_SWITCHED");
        filter.addAction("android.intent.action.USER_STOPPED");
        filter.addAction("android.intent.action.USER_UNLOCKED");
        mContext.registerReceiverAsUser(mUserChangeReceiver, UserHandle.ALL, filter, (String)null, (Handler)null);
    }
    private void unregisterReceiver() {
        mContext.unregisterReceiver(mUserChangeReceiver);
    }

// 注册与取消 用来监听用户切换的 某个 监听回调函数 listener
    public void registerOnUsersUpdateListener(UserProfileManager.OnUsersUpdateListener listener) {
        if (listener != null) {
            synchronized(mUpdateListeners) {
                if (mUpdateListeners.isEmpty())
                    registerReceiver(); // 操作的是: mUserChangeReceiver, 感觉还有点儿没有想通。。。。。。
                if (!mUpdateListeners.contains(listener))
                    mUpdateListeners.add(listener);
            }
        }
    }
    public void unregisterOnUsersUpdateListener(UserProfileManager.OnUsersUpdateListener listener) {
        synchronized(mUpdateListeners) {
            if (mUpdateListeners.contains(listener)) {
                mUpdateListeners.remove(listener);
                if (mUpdateListeners.isEmpty())
                    unregisterReceiver(); // 资源的释放
            }
        }
    }
}
```

7 lucid-profile-lib

- 这是一个 aidl 支持进程间通信的模块 module? 好像还不是很懂呀。。。。。。



```
~/ivi/carlibs/lucid-profile-lib/
<-> aidl
 <-> com
  <-> lucid
   <-> car
    <-> profiledata
     [+] IProfileDataService.aidl *
<+> build
<-> src
 <-> com
  <-> lucid
   <-> carlib
    <-> profile
     [+] DmsData.java *
     [+] Enrollment.java *
     [+] FaceAppearance.java *
     [+] FacialDetectionString.java *
     [+] LucidNavigationString.java *
     [+] ProfileAccountType.java *
     [+] ProfileData.java *
     [+] ProfileKey.java *
     [+] ProfileSwitcher.java *
     [+] ProfileSwitcherService.java *
     [+] UserProfileData.java *
[+] Android.mk *
[+] AndroidManifest.xml *
[+] build.gradle *
[+] settings.gradle *
```

```xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.lucid.carlib.profile">

    <uses-permission android:name="android.permission.INTERACT_ACROSS_USERS_FULL" />
    <uses-permission android:name="com.lucid.permission.RECEIVE_USERPROFILE_INTENTS"/>

    <application>

        <receiver
            android:name=".ProfileSwitcherService$Receiver"
            android:permission="com.lucid.permission.RECEIVE_USERPROFILE_INTENTS" >
            <intent-filter>
                <action android:name="com.lucid.intent.action.USER_LOGGED_IN" />
                <action android:name="com.lucid.intent.action.USER_LOGGED_OUT" />
            </intent-filter>
        </receiver>

        <service
            android:name=".ProfileSwitcherService"
            android:exported="false" />

    </application>

</manifest>
```

7.1 AndroidManifest.xml

```xml
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
          package="com.lucid.carlib.profile">
  <uses-permission android:name="android.permission.INTERACT_ACROSS_USERS_FULL"/> <!-- 系统级别的许可 -->
  <uses-permission android:name="com.lucid.permission.RECEIVE_USERPROFILE_INTENTS"/>
  <application>
    <receiver
        android:name=".ProfileSwitcherService$Receiver"
<!-- 这个不是前面 LucidUserProfile 应用中定义的广播许可吗？ -->
<!-- LucidCarService 应用中因为 run under headless User0, 也定义了这样一个许可 -->
        android:permission="com.lucid.permission.RECEIVE_USERPROFILE_INTENTS">
      <intent-filter>
        <action android:name="com.lucid.intent.action.USER_LOGGED_IN" />
        <action android:name="com.lucid.intent.action.USER_LOGGED_OUT" />
      </intent-filter>
    </receiver>
    <service
        android:name=".ProfileSwitcherService"
        android:exported="false" />
  </application>
</manifest>
```

- 这个跨进程模块监听 USER_LOGGED_IN 和 USER_LOGGED_OUT 广播，而这两个事件的发出者都是上面 LucidUserProfile 应用的 LucidDmsService.java 服务

7.2 LucidDmsService.java from LucidUserProfile App: 发送用户登录登出广播

```java
public class LucidDmsService extends Service {
    private BroadcastReceiver mReceiver = new BroadcastReceiver() {
        @Override
        public void onReceive(Context context, Intent intent) {
            Log.d(TAG, "On received HMI action: " + intent.getAction());

            if (ACTION_SCAN_DMS_PROFILE.equals(intent.getAction())) {
                mStateMachine.actionReceived(StateMachine.ACTION.REQUEST_SCAN, null);
            } else if (ACTION_RESET_DMS.equals(intent.getAction())) {
                mStateMachine.actionReceived(StateMachine.ACTION.REQUEST_RESET, null);
            } else if (ACTION_LOGOUT_DMS_PROFILE.equals(intent.getAction())) {
                mStateMachine.actionReceived(StateMachine.ACTION.REQUEST_LOGOUT, null);
            } else if (ACTION_DELETE_DMS_PROFILE.equals(intent.getAction())) {
                mStateMachine.actionReceived(StateMachine.ACTION.REQUEST_DELETE, intent.getExtras());
            } else if ( ACTION_DELETE_ALL_DMS_PROFILE.equals( intent.getAction() ) ) {
                mStateMachine.actionReceived(StateMachine.ACTION.REQUEST_DELETE_ALL, intent.getExtras());
            } else if ( ACTION_SHOW_ENROLLMENT_UI.equals( intent.getAction() ) ) {
                FaceScanUi.getInstance().showEnrollmentScreen();
            } else if ( ACTION_ENROLL_DMS_PROFILE.equals( intent.getAction() ) ) {
                mStateMachine.actionReceived(StateMachine.ACTION.REQUEST_ENROLL, intent.getExtras());
            } else if ( ACTION_REMOVE_ENROLLMENT_UI.equals( intent.getAction() ) ) {
                FaceScanUi.getInstance().removeView();
            } else if (ACTION_USER_UNLOCKED.equals(intent.getAction())) { // ACTION_USER_UNLOCKED: 发出者是: 系统?
                if (intent.hasExtra(EXTRA_USER_HANDLE)) {
                    int userId = intent.getIntExtra(EXTRA_USER_HANDLE, -1);
                    sendStickyBroadcastAsUser( new Intent(ACTION_READ_BLUETOOTH_INFO), UserHandle.of(userId) );
                    Intent userProfileIntent = new Intent(ACTION_USER_LOGGED_IN); // 发送 ACTION_USER_LOGGED_IN 广播
                    userProfileIntent.putExtra(EXTRA_USER_ID, userId);
                    sendBroadcastAsUser( userProfileIntent, UserHandle.of(userId), LUCID_APPS_PERMISSION  );
                    Log.d(TAG,"Got a handle for user: "+ userId);
                    if (ProfileLauncher.getFTEBoolean(context))
                        notifyLucidAppsForProfileCreation();
                }
                // 。。。
            } else if (ACTION_USER_STOPPED.equals(intent.getAction())) {
                if (intent.hasExtra(EXTRA_USER_HANDLE)){
                    int userId = intent.getIntExtra(EXTRA_USER_HANDLE,-1);
                    Intent userProfileIntent = new Intent(ACTION_USER_LOGGED_OUT); // 发送 ACTION_USER_LOGGED_OUT 广播
                    userProfileIntent.putExtra(EXTRA_USER_ID, userId);
                    sendBroadcastAsUser( userProfileIntent, UserHandle.of(userId), LUCID_APPS_PERMISSION );
                    Log.d(TAG,"Got a handle for user: "+ userId);
                }
            } else if (ACTION_UPDATE_USER_INFO.equals(intent.getAction())) {
               // ...
        }
    }
}
```

- 这两个事件的广播监听是需要许可的,所以需要用户定义的许可 com.lucid.permission.RECEIVE_US
- 这个许可是由广播的发出者应用 LucidUserProfile 定义的

```
<permission
    android:name="com.lucid.permission.RECEIVE_USERPROFILE_INTENTS"
    android:label="lucid_userprofile_permission"
    android:protectionLevel="signature" />
```

7.3 ProfileSwitcher.java: 抽象类，定义了两个抽象方法，供应用层各应用去实现，从而实现用户登录登出的全局监听回调，相关应用可以监听据此去实现它们自己的个性化配置操作

```
public abstract class ProfileSwitcher {
    // Notification API for profile switching.
    public abstract void onProfileLoggedIn(Context context, int userId);
    public abstract void onProfileLoggedOut(Context context, int userId);

    //=============================================================
    // Persist & Retrieve Profile Data.
    //   Implementation can be further updated in future
    //   (e.g. Syncing to the cloud...)
    //   Locally caching to SharedPreferences for now.
    //=============================================================
    private static final String LOCAL_PROFILE_CACHE = "LOCAL_PROFILE_CACHE";
    private void persistProfileData(Context context, Bundle profileData) {
        SharedPreferences.Editor editor = context.getSharedPreferences(
            LOCAL_PROFILE_CACHE, Context.MODE_PRIVATE ).edit();
        persistBundle( editor, LOCAL_PROFILE_CACHE, profileData );
        editor.apply();
    }
    private Bundle retrieveProfileData(Context context) {
        SharedPreferences sharedPref = context.getSharedPreferences(
            LOCAL_PROFILE_CACHE, Context.MODE_PRIVATE );
        return retrieveBundle( sharedPref, LOCAL_PROFILE_CACHE );
    }
    private static final String SAVED_PREFS_BUNDLE_KEY_SEPARATOR = "ğğ";
    private static void persistBundle(SharedPreferences.Editor editor, String key, Bundle dataBundle) {
        // ...
    }
    private static Bundle retrieveBundle(SharedPreferences sharedPreferences, String key) {
        // ...
    }
}
```

7.4 ProfileSwitcherService.java: 代码基本理解，需要再想一下整个安卓系统应用层用户切换时的处理过程

- 这个服务充当一个桥梁，搭建应用层其它应用在用户登录登出时方便操作些什么的入口路径

```
public class ProfileSwitcherService extends IntentService {
    private static final String TAG = "UserProfileClientService";
    private static final String PROFILE_SWITCH_HANDLER_CLASS = "ProfileSwitchHandlerClass";

    private static final String ACTION_USER_LOGGED_IN = "com.lucid.intent.action.USER_LOGGED_IN";
    private static final String ACTION_USER_LOGGED_OUT = "com.lucid.intent.action.USER_LOGGED_OUT";
    private static final String EXTRA_USER_ID = "com.lucid.intent.extra.USER_ID";
//==================================================================
// Receiver
//==================================================================
    public static class Receiver extends BroadcastReceiver {
        @Override
            public void onReceive(Context context, Intent intent) {
            if ( intent == null )
                return;
            final String action = intent.getAction();
            int userId = intent.getIntExtra( EXTRA_USER_ID, -1 );
            Intent serviceIntent = new Intent( context, ProfileSwitcherService.class );
            serviceIntent.putExtra( EXTRA_USER_ID, userId );
            if ( ACTION_USER_LOGGED_IN.equals( action ) ) {
                serviceIntent.setAction( ACTION_USER_LOGGED_IN );
                context.startService( serviceIntent );
            } else if ( ACTION_USER_LOGGED_OUT.equals( action ) ) {
                serviceIntent.setAction( ACTION_USER_LOGGED_OUT );
                context.startService( serviceIntent );
            }
        }
    }
//==================================================================
// Service
//==================================================================
    public ProfileSwitcherService() {
        super( TAG );
    }
    @Override protected void onHandleIntent(Intent intent) {
        if ( intent == null ) return;
        // 通过反射的方法拿到其它应用的调用类相关信息，方便调用回调 ??
        ProfileSwitcher profileSwitcher = null;
        try {
            ApplicationInfo appInfo = getPackageManager()
                .getApplicationInfo( getPackageName(), PackageManager.GET_META_DATA );
            if(appInfo!=null) {
                String switchHandlerClazzName = appInfo.metaData
                    .getCharSequence( PROFILE_SWITCH_HANDLER_CLASS ).toString();
                Log.d(TAG, "switchHandlerClazzName: " + switchHandlerClazzName);
                Class<?> switchHandlerClazz = Class.forName(switchHandlerClazzName);
                Constructor<?> creator = switchHandlerClazz.getConstructor();
                Object object = creator.newInstance();
                if (object instanceof ProfileSwitcher)
                    profileSwitcher = (ProfileSwitcher) object;
            }
        } catch ( Exception e ) {
            return;
        }
        if ( profileSwitcher == null ) return;
        final String action = intent.getAction();
        int userId = intent.getIntExtra( EXTRA_USER_ID, -1 );
        if ( ACTION_USER_LOGGED_IN.equals( action ) ) {
            Log.d( TAG, "Start profile logged in : userId=" + userId );
            profileSwitcher.onProfileLoggedIn( getApplicationContext(), userId );
```

```
                Log.d( TAG, "Finish profile logged in : userId=" + userId );
            } else if ( ACTION_USER_LOGGED_OUT.equals( action ) ) {
                Log.d( TAG, "Start profile logged out : userId=" + userId );
                profileSwitcher.onProfileLoggedOut( getApplicationContext(), userId );
                Log.d( TAG, "Finish profile logged out : userId=" + userId );
            }
        }
    }
```

7.5 IProfileDataService.java: AIDL 定义的四个接口方法：主要用于用户账户的 CRUD 等操作？

```
package com.lucid.car.profiledata;

interface IProfileDataService {

    void persistProfileData(String userID, int profileType, String profileKey, String profileData, boolean syncToCloud);
    String retrieveProfileData(String userID, int profileType, String profileKey, String defaultValue);
    void deleteProfileData(String userID, int profileType,  String profileKey, boolean syncToCloud);
    void deleteProfile(String userID, boolean syncToCloud);
}
```

7.6 ProfileData.java: 进程间通信，相对复杂一点儿

- 前面讲到，这个模块在系统中是可以进程间通信的，那么原理是怎样的呢？先从这个类的 comment 从别人的注解来理解一下

```
This version of ProfileData (as of 27 APR 2021) is a step toward objectifying access to
persist/retrieve methods, which up till now have been implemented only as static methods.

To use ProfileData it *must first be instantiated*, e.g.:
      profileData = new ProfileData( getContext() );
 The binding to ProfileDataService takes place upon instantiation, within the constructor.

Once instantiated, methods can be access via the object, e.g.:
     value = profileData.retrieveData( context, key);
Also, after instantiation and for backwards-compatibility until such time they are
  deprecated, existing static objects via class reference and using the same method names,  e.g.:
     value = ProfileData.retrieveProfileData( context, key);

Two additional methods were added to check the connection to the service:
     public static boolean isBoundToPTLService()
     public boolean isBoundToProfileDataService()
These methods check to see if a value has been assigned to the service connection
object (connected/bound, returns true) or if it is null (disconnected/unbound, returns false).

At an appropriate time in the future, all static methods will be removed from this class.
```

- 主要是与远程服务之间的联接与断开的通信处理?

```
public class ProfileData {
    private static final String TAG = "ProfileData";
    private final Context mContext;
    private IProfileDataService mProfileDataService = null;
    private ServiceConnectionListener mServiceConnectionListener = null;
    private final boolean syncToCloudDefault = true;
    private static final int SERVICE_REQUEST_TYPE_PERSIST = 1;
    private static final int SERVICE_REQUEST_TYPE_RETRIEVE = 2;
    private static final int SERVICE_REQUEST_TYPE_UPDATE = 3;
    private static final int SERVICE_REQUEST_TYPE_DELETE = 4;
    private static final String NO_DEFAULT_VALUE_REQUIRED = null;
    private boolean mIsServiceBound;
//*********************************************************************************************
// Constructors
//*********************************************************************************************
    public ProfileData(Context context) { //
        Log.d(TAG, "ProfileData() ");
        mContext = context.getApplicationContext();
    }
//*********************************************************************************************
```

```
// Service connection management methods
//*******************************************************************************************
// 谁在实现这个接口: LucidCarService 应用中的 LucidProfileSessionService.java 实现和实例化了这个接口
// 谁在实现这个接口? 所有想要使用 ProfileData instance 实例的类与服务, 都需要实现接个接口类
    public interface ServiceConnectionListener {
        void onConnected();
        void onDisconnected();
    }
    public void bindService(ServiceConnectionListener listener) { // 需要考虑当用户登出时, 需要做哪些工作
        Log.d(TAG, "bindService() ");
        String m_TAG = "bindService(listener) bindToProfileDataService(): ";
        // Stopwatch bindStopwatch = new Stopwatch("com.lucid.profile.binding", 0.1);
        if ( isServiceBound()) {
            Log.w( TAG, m_TAG + " Service already bound...");
            return;
        }
        Log.i( TAG, m_TAG + "Binding ProfileDataService...");
        mServiceConnectionListener = listener;
        mIsServiceBound =  mContext.bindServiceAsUser( getServiceIntent(), mServiceConnection,
                                                       Context.BIND_AUTO_CREATE, UserHandle.SYSTEM );
        if (Feature.DEVELOPER_BUILD || bindStopwatch.check()) { //
            System.err.println(bindStopwatch.log());
            bindStopwatch.reset();
        }
    }
    public void unbindService() {
        Log.d(TAG, "unbindService() ");
        String m_TAG = "ProfileData unbindService(): ";
        Stopwatch unbindStopWatch = new Stopwatch("com.lucid.profile.unbinding", 0.1);
        if ( mIsServiceBound ) {
            Log.i( TAG, m_TAG + "Unbinding ProfileDataService...");
            mContext.unbindService( mServiceConnection );
            mIsServiceBound = false;
        }
        mServiceConnectionListener = null;
        if ( Feature.DEVELOPER_BUILD || unbindStopWatch.check() ) {
            System.err.println(unbindStopWatch.log());
            unbindStopWatch.reset();
        }
    }
    private final ServiceConnection mServiceConnection = new ServiceConnection() {
        @Override
        public void onServiceConnected(ComponentName name, IBinder service) {
            Log.d(TAG, "onServiceConnected() IPC");
            String m_TAG = "onServiceConnected(): ";
            Log.i(TAG, m_TAG + "bound to service" );
            if ( Feature.DEVELOPER_BUILD || mConnectStopwatch.check()) {
                System.err.println( mConnectStopwatch.log() );
                mConnectStopwatch.reset();
            }
            mProfileDataService = IProfileDataService.Stub.asInterface( service ); // 建立联接后, 赋值
            if ( mServiceConnectionListener != null )
                mServiceConnectionListener.onConnected();
        }
        @Override
        public void onServiceDisconnected( ComponentName name ) {
            Log.d(TAG, "onServiceDisconnected() IPC");
            String m_TAG = "onServiceDisconnected(): ";
            Log.i(TAG, m_TAG + "unbound from service");
            if ( Feature.DEVELOPER_BUILD || mDisconnectStopwatch.check() ) {
                System.err.println( mDisconnectStopwatch.log() );
                mDisconnectStopwatch.reset();
            }
            if ( mServiceConnectionListener != null )
                mServiceConnectionListener.onDisconnected();
            mProfileDataService = null; // 断开联接后, 重新设置值
        }
    };
    public boolean isServiceBound() { // 本地的简单判断方法
        return ( mProfileDataService != null );
    }
    private static final String DEFAULT_PROFILE_DATA_SERVICE_PACKAGE = "com.lucid.car.profiledata";
    private static final String DEFAULT_PROFILE_DATA_SERVICE_CLASS = "com.lucid.car.profiledata.ProfileDataService";
    private Intent getServiceIntent() {
        return (new Intent().setComponent(new ComponentName(
                                                DEFAULT_PROFILE_DATA_SERVICE_PACKAGE,
                                                DEFAULT_PROFILE_DATA_SERVICE_CLASS ) ) );
    }
//*********************************************************************************************
// Retrieve methods
//*********************************************************************************************
    public String retrieveString(String key) {}
    public String retrieveString(String key, String defaultValue) {}
    public Float retrieveFloat(String key) {}
    public Float retrieveFloat(String key, float defaultValue) {}
    public Integer retrieveInt(String key) {}
    public Integer retrieveInt(String key, int defaultValue) {}
    public Boolean retrieveBoolean(String key) {}
    public Boolean retrieveBoolean(String key, boolean defaultValue) {}
//*********************************************************************************************
// Persist methods
//*********************************************************************************************
    public void persistString(String key, String value) {}
    public void persistInt(String key, int value) {}
    public void persistFloat(String key, float value) {}
    public void persistBoolean(String key, boolean value) {}
//*********************************************************************************************
// Delete methods
//*********************************************************************************************
    public void deleteProfile( String userId, boolean syncToCloud ) {    }
//=============================================================================================
// Exception handling support methods
//=============================================================================================
    private static void handleAIDLException(String m_TAG, String key, String userID, String value, int serviceRequestType,
        Log.d(TAG, "handleAIDLException() ");
        String m_message = m_TAG + "Error occurred while making AIDL call to ";
        switch ( serviceRequestType ) {
        case SERVICE_REQUEST_TYPE_PERSIST:
            m_message = m_message + "persist data for key= " + key + ", value= " + value;
            break;
        case SERVICE_REQUEST_TYPE_RETRIEVE:
            m_message = m_message + "retrieve data for key= " + key;
            break;
        case SERVICE_REQUEST_TYPE_DELETE:
            if( userID != null )
                m_message = m_message + "delete profile for userID= " + userID;
            else
                m_message = m_message + "delete data for key= " + key;
            break;
        default:
            break;
        }
        Log.e( TAG, m_message + ", exception info follows: " + e.getMessage() );
    }
    private static void handleNumberFormatException(String m_TAG, String key, String value, Exception e) {
        Log.e( TAG, m_TAG + "Error occurred during conversion, retrieved string value= " + value + ", exception info follow
    }
//*********************************************************************************************
// Deprecated static api access
//*********************************************************************************************
// Singleton instance.
    private static ProfileData sInstance = null;
    @Deprecated
    private static ProfileData getInstance(Context context) {
        Log.d(TAG, "getInstance() ");
        if (sInstance == null) {
            synchronized (ProfileData.class) {
                if (sInstance == null) {
                    sInstance = new ProfileData(context);
                    sInstance.bindService( null );
                }
            }
        }
        return sInstance;
    }
}
```

7.7 Beans: 因为涉及到数据库操作？进程通信，应用中还有一堆 beans, 暂且视而不见吧。。。

8 LucidProfileSetup：这个应用，在用户初始化设置之后，很多地方都是可以跳过不用的

- 应用有多处来源的监听：监听 fte_step（<action android:name="fte_step"/>）。自己应用的 CIDScreenProfileSetup.java 发送的只是本地广播"ftestep", 也是"fte_step".
- 监听系统 BOOT_COMPLETED 事件，设置用户账户: 从 Figma 的界面来看，有很多内容，这里暂时不细看，只在意把应用间的逻辑先联贯起来

8.1 AndroidManifest.xml

```
<application
    android:allowBackup="true"
    android:icon="@mipmap/ic_launcher"
    android:label="@string/app_name"
    android:roundIcon="@mipmap/ic_launcher_round"
    android:supportsRtl="true"
    android:theme="@style/appTheme" >
  <activity android:name="com.lucid.profilesetup.view.ProfileSetupHostActivity"
            android:exported="true"
            android:launchMode="singleTop">
  </activity>
  <activity android:name="com.lucid.profilesetup.view.CIDScreenProfileSetup"
            android:exported="true"
            android:launchMode="singleTop">
  </activity >

<!-- 是否可以实例化服务。如果可以，则设为 “true”; 如果不能，则设为 “false”。默认值为 “true”  -->
  <service android:name=".support.FRSignalController.FRSignalControlerService"
           android:enabled="true"/>
  <receiver android:name=".view.FteStepsBroadcastReceiver">
    <intent-filter>
      <category android:name="android.intent.category.DEFAULT" />
      <action android:name="fte_step" />
    </intent-filter>
  </receiver>
  <receiver
      android:name=".support.FRSignalController.OnBootReceiver"
      android:enabled="true"
      android:exported="true">
    <intent-filter>
      <action android:name="android.intent.action.BOOT_COMPLETED" />
    </intent-filter>
  </receiver>
</application>
```

8.2 OnBootReceiver.java: 应用入口，通过接收广播切入

```
public class OnBootReceiver extends BroadcastReceiver {
    private static final String TAG = "OnBootReceiver";
    @Override public void onReceive(Context context, Intent intent) {
        String FILENAME = context.getFilesDir()+"/created_on_first_bootup";
        Log.d(TAG, FILENAME);
        UserManager um = (UserManager) context.getSystemService(Context.USER_SERVICE);
        Log.d(TAG, "isSystemUser: " + Boolean.toString(um.isSystemUser()));
        if (um.isSystemUser()) {
            // The file /data/created_on_first_bootup creaed on first time bootup.
            File file = new File(FILENAME);
            if (!file.exists()) {
                Log.d(TAG, "it is first time boot up sending FR signal IDLE");
                //Send FR signal IDLE
                Intent serviceIntent = new Intent(context, FRSignalControlerService.class);
                context.startService(serviceIntent);
                if (!file.exists())
```

```
                    file.mkdirs();
            }

        }
    }
}
```

8.3 FRSignalDispatcher.java：这个跟一般 Controller 的写法比较像，稍微看一下就可以了

- 当给 VCU 重置了 1/0 中的某个状态，是会触发 PowerState 的状态改变吗？接下来的逻辑仍然是连不起来。。。。。。

```java
public class FRSignalDispatcher {
    private static final String TAG = "FRSignalDispatcher";

    public static final boolean IS_DEBUG_BUILD = "userdebug".equals(Build.TYPE) || "eng".equals(Build.TYPE);
    public static final int IDLE_STATE = 0;
    public static final int RESET_STATE = 1;
    private Context mContext;
    private FRSignalResponseListener mFRSignalResponseListener;

    public static interface FRSignalResponseListener { // Notification callback interface...
        public void onVhalConnectedW();
        public void onVhalDisconnectedW();
        public void onVCUPropertiesReadyW(); // 这里是不是跟 PowerStates 相关的硬件又联系起来了呢？
        public void onVehicleGearStateChanged(int gearState);
    }
    public enum SignalType {
        IDLE,
        RESET,
    }
    //=================================================================
    // Init / Cleanup
    //=================================================================
    public FRSignalDispatcher() {
        mHandlerThread = new HandlerThread(TAG);
        mHandlerThread.start();
        mSignalBroadcastHandler = new FRSignalDispatcher.SignalBroadcastHandler(Looper.getMainLooper());
    }
    public void init(Context context, FRSignalResponseListener signalResponseListener) {
        mContext = context;
        mFRSignalResponseListener = signalResponseListener;
        mCarApiClient = Car.createCar(mContext, mCarConnectionCallback);
        mCarApiClient.connect();
    }
    public void cleanup() {
        if (mCarApiClient != null)
            mCarApiClient.disconnect();
        mFRSignalResponseListener = null;
        mContext = null;
    }

    //=================================================================
    // Connection to VHal
    //=================================================================
    private static final int VPROP_ID_VCU_GEAR_STATE = 0x000; // yet to define
    private final static int[] S_MONITOR_VPROP_IDS = new int [] {
        VPROP_ID_VCU_GEAR_STATE,
    };
    private Car mCarApiClient;
    private CarPropertyManager mPropertiesManager;
    private ValuesDumper dumper;

    //=================================================================
    // DataStore
    //=================================================================
    @GuardedBy("mVehicleGearState")
    private Integer mVehicleGearState = 0;
    private final ServiceConnection mCarConnectionCallback = new ServiceConnection() {
        @Override public void onServiceConnected(ComponentName name, IBinder service) {
            if (mFRSignalResponseListener != null)  // Notify
```

```
                mFRSignalResponseListener.onVhalConnectedW();
            try {
                mPropertiesManager = (CarPropertyManager) mCarApiClient
                    .getCarManager(android.car.Car.PROPERTY_SERVICE);
                for (int vPropId : S_MONITOR_VPROP_IDS) {
                    mPropertiesManager.registerCallback(mPropertiesListener, vPropId, 1);
                }
                // Fetch initial values.
                int gearState = mPropertiesManager.getIntProperty(VPROP_ID_VCU_GEAR_STATE, 0);
                synchronized (mPropertiesLock) {
                    mVehicleGearState = gearState;
                }
            } catch (CarNotConnectedException e) {
                Log.e(TAG, "Car not connected in onServiceConnected");
            }
            if (mFRSignalResponseListener != null)  // Notify
                mFRSignalResponseListener.onVCUPropertiesReadyW();
            if ( IS_DEBUG_BUILD ) {
                //dumper = new ValuesDumper(TAG, S_MONITOR_VPROP_IDS, mSignalBroadcastHandler, mPropertiesManager);
            }
        }
        @Override
        public void onServiceDisconnected(ComponentName name) {
            if( IS_DEBUG_BUILD ) {
                dumper.stop();
                dumper = null;
            }
            for (int vPropId : S_MONITOR_VPROP_IDS) {
                mPropertiesManager.unregisterCallback(mPropertiesListener, vPropId);
            }
            if (mFRSignalResponseListener != null) {
                mFRSignalResponseListener.onVhalDisconnectedW();
            }
            // Retry connection again in some interval.
        }
    };

    //=================================================================
    // Handling Broadcasting of Signals (from CCC)
    //=================================================================
    private static final int MSG_BROADCAST_SIGNAL = 100;
    private HandlerThread mHandlerThread;

    private FRSignalDispatcher.SignalBroadcastHandler mSignalBroadcastHandler;
    class SignalBroadcastHandler extends Handler {
        SignalBroadcastHandler(Looper looper) {
            super(looper);
        }
        @Override
            public void handleMessage(Message msg) {
            Log.d(TAG, ">>handleMessage");
            if (msg.what == MSG_BROADCAST_SIGNAL) {
                broadcastCccSignalW(msg.arg1, msg.arg2);
            }
        }
        void broadcastCccSignalW(int propId, int value) {
            Log.d(TAG, "broadcastCccSignalW");
            try {
                mPropertiesManager.setIntProperty(propId, 0, value);
            } catch (android.car.CarNotConnectedException e) {
                Log.e(TAG, "broadcastCccSignalW() : VHAL not connected.");
            }
        }
    }

    //=================================================================
    // Car Property IDs
    //=================================================================
    private static final int FACTORY_RESET_INIT = 0x21401E00; // Factory Reset Idle:0, Reset:1

    //=================================================================
    // VHAL Properties
    //=================================================================
    private CarPropertyManager.CarPropertyEventCallback mPropertiesListener =
        new CarPropertyManager.CarPropertyEventCallback() {
```

```
        @Override public void onChangeEvent(CarPropertyValue carPropertyValue) {
            switch (carPropertyValue.getPropertyId()) {
            case VPROP_ID_VCU_GEAR_STATE:
            handleVehicleGearStateChanged((Integer) carPropertyValue.getValue());
            break;
            }
        }
        @Override public void onErrorEvent(int i, int i1) {
            Log.e(TAG, "onErrorEvent: i=" + i + ", i1=" + i1);
        }
    };

    //================================================================
    // Vehicle VCM State
    //================================================================

    public static final int VEHICLE_GEAR_STATE_UNDEFINED = 0;
    public static final int VEHICLE_GEAR_STATE_UNLOCK_MODE = 1;
    public static final int VEHICLE_GEAR_STATE_LOCK_MODE = 2;

    private final Object mPropertiesLock = new Object();

    private void handleVehicleGearStateChanged(int gearState) {
        synchronized (mPropertiesLock) {
            mVehicleGearState = gearState;
        }
        if (mFRSignalResponseListener != null) {
            mFRSignalResponseListener.onVehicleGearStateChanged(gearState);
        }
    }
    /**
     * @param type Idle: {@value #IDLE_STATE}
     *             Off: {@value #RESET_STATE}
     *             Send signal to TCU
     **/ // 当给 VCU 重置了 1/0 中的某个状态，是会触发 PowerState 的状态改变吗？接下来的逻辑仍然是连不起来。。。。。
    public void sendFactoryResetSignal(SignalType type) {
        Log.d(TAG, "sendFactoryResetSignal");
        switch (type) {
        case IDLE:
            mSignalBroadcastHandler.sendMessage(mSignalBroadcastHandler
                                                .obtainMessage(MSG_BROADCAST_SIGNAL, FACTORY_RESET_INIT, IDLE_STATE));
            break;
        case RESET:
            mSignalBroadcastHandler.sendMessage(mSignalBroadcastHandler
                                                .obtainMessage(MSG_BROADCAST_SIGNAL, FACTORY_RESET_INIT, RESET_STATE));
            break;
        }
    }
}
```

8.4 FRSignalControlerService.java: 回字的四样写法 IBinder vs Daemont/Client contoller 区别是 ?????

- 体会回字的四样写法：写成服务（IBinder 机制）的形式与 Controller DaemonClient 形式有什么区别呢?

```
public class FRSignalControlerService extends Service {
    static private final String TAG = "FRSignalControlerService";

    private FRSignalDispatcher mFRSignalDispatcher;
    final Messenger mFRSigCtrMessenger = new Messenger(new IncomingHandler()); // 定义在下面
    @Override
        public int onStartCommand(Intent intent, int flags, int startId) {
        init( new FRSignalDispatcher.FRSignalResponseListener() { // static interface 接口里定义的方法实现
                @Override public void onVhalConnectedW() {
                    Log.d(TAG, ">> Send FR signal IDLE_STATE"); // CCC 向下向硬件发信号
                    mFRSignalDispatcher.sendFactoryResetSignal(FRSignalDispatcher.SignalType.IDLE);
                }
                @Override public void onVhalDisconnectedW() {  }
                @Override public void onVCUPropertiesReadyW() {  }
                @Override public void onVehicleGearStateChanged(int gearState) {  }
            });
        return START_NOT_STICKY; //
    }
    @Override
        public IBinder onBind(Intent intent) {
        Log.d(TAG, "------onBind-------");
        init( new FRSignalDispatcher.FRSignalResponseListener() {
                @Override public void onVhalConnectedW() {
                    Log.d(TAG, "onVhalConnectedW");
                    Log.d(TAG, ">> Send FR signal RESET_STATE");
                    mFRSignalDispatcher.sendFactoryResetSignal(FRSignalDispatcher.SignalType.RESET);
                }
                @Override public void onVhalDisconnectedW() {  }
                @Override public void onVCUPropertiesReadyW() {  }
                @Override public void onVehicleGearStateChanged(int gearState) {  }
            });
        return mFRSigCtrMessenger.getBinder();
    }
    public void init(FRSignalDispatcher.FRSignalResponseListener frSignalResponseListener) {
        Context context = getApplicationContext();
        Log.d(TAG, "FRSignalController is null =" + (mFRSignalDispatcher == null));
        if (context != null) {
            if (mFRSignalDispatcher == null)
                mFRSignalDispatcher = new FRSignalDispatcher();
            mFRSignalDispatcher.init(context.getApplicationContext(), frSignalResponseListener);
        }
    }
    class IncomingHandler extends Handler {
        @Override
            public void handleMessage(Message msg) {
            switch (msg.what){
            case FRSignalDispatcher.IDLE_STATE:
                mFRSignalDispatcher.sendFactoryResetSignal(FRSignalDispatcher.SignalType.IDLE);
                break;
            case FRSignalDispatcher.RESET_STATE:
                mFRSignalDispatcher.sendFactoryResetSignal(FRSignalDispatcher.SignalType.RESET);
                break;
            default:
                break;
            }
        }
    }
    public void cleanup() { // 资源回收: Controller 的回收
        if (mFRSignalDispatcher != null) {
            Log.d(TAG, "cleanup: calling Controller.cleanup()");
            mFRSignalDispatcher.cleanup();
            mFRSignalDispatcher = null;
        }
    }
    @Override public void onDestroy() {
        cleanup();
        super.onDestroy();
    }
    @Override public void onCreate() {
        super.onCreate();
    }
}
```

8.5 CIDScreenProfileSetup.java

8.5.1 发送本地广播 “ftestep”

```
private void sendLocalBroadcast(FteScreenFlow.FteSteps ftestate) {
    Log.d(TAG, "send local fte step broadcast message");
// 这一行, 非常明确地指出了过滤的是 FteStepsBroadcastReceiver.FTE_STEP_INFO, 也就是 "fte_step"
    Intent i = new Intent(FteStepsBroadcastReceiver.FTE_STEP_INFO);
    i.putExtra("ftestep", ftestate.getValue());
    LocalBroadcastManager.getInstance(this).sendBroadcast(i);
}
```

8.5.2 IProfileDataService AIDL 跨进程服务的服务绑定

- 这里着重去看：当它设置 mProfileData 的时候，它就需要实现 lucid-profile-lib 中 ProfileData.java 所定义的公共类里的所有回调接口方法
- 这里把这个公用接口再贴一下

```
public interface ServiceConnectionListener { // 谁在实现这个接口? 所有想要用 ProfileData instance 实例的类
    void onConnected();
    void onDisconnected();
}
```

- 下面贴剪接的源码

```
// <activity android:name="com.lucid.profilesetup.view.CIDScreenProfileSetup"
//      android:exported="true" android:launchMode="singleTop">
// </activity >
public class CIDScreenProfileSetup extends AppCompatActivity {
    private static final String TAG = CIDScreenProfileSetup.class.getName();

    private ProfileData mProfileData;

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

        mProfileData = new ProfileData(getApplicationContext());
        mProfileData.bindService(new ProfileData.ServiceConnectionListener() {
                @Override public void onConnected() {
// UserProfileData:  lucid-profile-lib 中的一个类，所有成员变量与成员方法均为 static，用以表示任何时候， lucid air 前台有且只有唯
                    UserProfileData.setProfileDataInstance(mProfileData);
                    // checkAndFlagAlreadyConfiguredFteSteps(); // 这些者暂且不作关注
                    // mIcrScreenProfileSetup.showView();
                    // mViewModel.startFirstFteScreen( getApplicationContext() );
                }
                @Override public void onDisconnected() {
                    UserProfileData.setProfileDataInstance(null);
                }
            });
    }
    public void onEventReceived(FteScreenFlow.FteSteps step) {
        Log.d(TAG, "onEventReceived");
        if (FteScreenFlow.FteSteps.FTE_EXIT == step) {
// 当用户注册完成之后，启动的是 LucidSystemUI 应用中的 LucidSysUiService.java 服务
            mIcrScreenProfileSetup.startSystemUIService();
            mIcrScreenProfileSetup.cleanup();
            finish();
            return;
        }
// 。。。。
    }
    @Override public void onDestroy() {
        if (mProfileData.isServiceBound())
            mProfileData.unbindService();
        mProfileData = null;
        UserProfileData.setProfileDataInstance(null);
        super.onDestroy();
    }
    // private void checkAndFlagAlreadyConfiguredFteSteps(){
    //     int value = Settings.Global.getInt(
    //         getContentResolver(), TAC_STATE_KEY, TAC_INIT);
    //     mViewModel.setIsNavLiteTnCDone(value != TAC_INIT);
    //     String vin = VehicleInfoSettings.getVINNumber(getApplicationContext().getContentResolver());
    //     mViewModel.setIsProfileHasDefaultPin(vin);
    // }
}
```

8.6 FRSignalControlerService.java

- 接收本地广播 “fte_step”

```
<receiver android:name=".view.FteStepsBroadcastReceiver">
  <intent-filter>
```

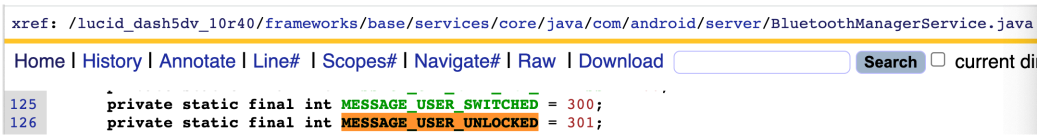
```
        <category android:name="android.intent.category.DEFAULT" />
        <action android:name="fte_step" />
    </intent-filter>
</receiver>
```

- 这个接收器接收到本应用本地 fte_step 广播后，调用自己定义的 FteEvent 回调接口

```
public class FteStepsBroadcastReceiver extends BroadcastReceiver {
    private static final String TAG = "FteBroadCastReceiver";

    public static final String FTE_STEP_INFO = "fte_step"; //
    private FteEvent mFteEventCallBack;

    public FteStepsBroadcastReceiver(FteEvent fteEvent) {
        mFteEventCallBack = fteEvent;
    }
    @Override
        public void onReceive(Context context, Intent intent) {
        FteScreenFlow.FteSteps fteStep = FteScreenFlow.FteSteps.valueOf(intent.getIntExtra("ftestep", 9));
        mFteEventCallBack.onReceived(fteStep);
    }
    public interface FteEvent {
        void onReceived(FteScreenFlow.FteSteps fteSteps);
    }
}
```

8.7 MESSAGE_USER_UNLOCKED: 这是哪里用到的呢 ???

- 定义的地方


xref: /lucid_dash5dv_10r40/frameworks/base/services/core/java/com/android/server/BluetoothManagerService.java

Home | History | Annotate | Line# | Scopes# | Navigate# | Raw | Download
Search
current di

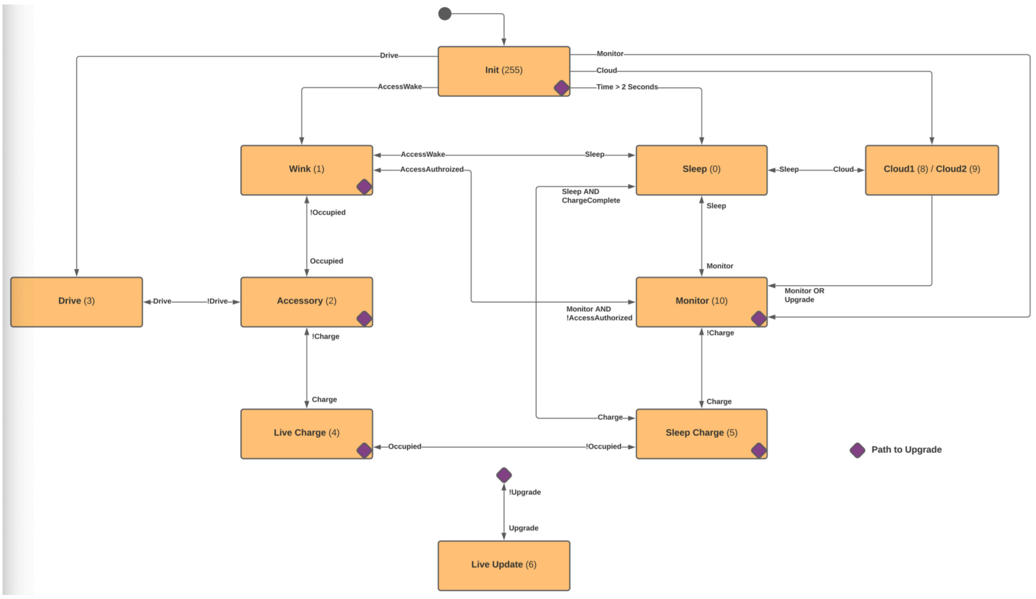
```
        private static final int MESSAGE_USER_SWITCHED = 300;
        private static final int MESSAGE_USER_UNLOCKED = 301;
```

9 Android 10 + Lucid : Power State Management

- Vehicle Power States Single Point of Truth: https://lucidmotors.atlassian.net/wiki/spaces/BCU/pages/1129480648/Vehicle+Power+States
- Paul's page: https://lucidmotors.atlassian.net/wiki/spaces/INFO/pages/1685230984/Profile+-+Vehicle+Power+States
- Description of States:
  - (0) SLEEP - Vehicle is powered off or in the process of powering down
  - (1) WINK - Vehicle is powered on or in the process of powering on, cabin is unoccupied but access device is detected
  - (2) ACCESSORY - Cabin is occupied, all 12V devices are available
  - (3) DRIVE - Gear is Drive or Reverse, vehicle can produce torque
  - (4) LIVE CHARGE - Charger is connected, access device is detected or cabin is occupied
  - (5) SLEEP CHARGE - Charger is connected, access device not detected and cabin unoccupied
  - (8) CLOUD1 - BMU has woken up the vehicle to update data in the cloud, QM rail not powered

    - (9) CLOUD2 - TCU has woken up the vehicle to update data in the cloud, QM rail not powered
    - (10) MONITOR - Vehicle is unoccupied, no access device detected, QM rail powered on for one of the reasons below:
        * HVAC conditioning
        * Battery conditioning
        * Alarm
        * TCU Monitor request
    - (6) LIVE UPDATE- Software update for 1 or more ECUs, QM rail powered on
    - (255) Init - Initial value of vehicle power state until valid state is determined

- State Diagram:



- State Transitions: 这里需要再整理一下，改天再整理

Transition Description AccessWake Access Control System is requesting wake Occupied Cabin is detected to be occupied AccessAuthorized Key is detected Drive Gear position is Drive or Reverse Charge Charge cable is connected ChargeComplete Charging is complete Monitor TCU Monitor State Request BMU Monitor State Request Alarm is active Hazard lights active and not Key Detected HVAC Pre-conditioning Cloud Cloud1 - BMU Cloud State Request Cloud2 - TCU Cloud State Request Upgrade TCU Request for Upgrade Vehicle unoccupied VCU Indicates safe to upgrade Hazard lights NOT active Sleep NOT Drive condition NOT AccessWake condition NOT Cloud condition NOT Monitor condition Doors closed Trunk/Frunk closed

10 Profiles in Android 10

- https://lucidmotors.atlassian.net/wiki/spaces/INFO/pages/2737014059/Profiles+in+Android+10

10.1 Use Cases:

- Lucid profile app supports creating multiple users on a single car by separating user accounts and application data. For instance, Primary drier may allow their family & friends can share a car by separating user accounts and application data.
- Each profile account holds the individual personalized configuration stored like lucid a/c, Seats & steering configurations , HVAC settings etc.
- Profile switches automatically from different inputs like FaceScan, keyfob link, mobile key etc. or manually switch from status bar (*1).

10.2 Categories of users:

- Lucid profiles uses the following categories of users.
  - **System user** Is a headless user created to a device. The system user cannot be removed except by factory reset, this is a user always running in the background even when other User Profiles are in the foreground.
  - **Primary Driver** A user who has the permission to create and remove other users, as well as control some general multi-user settings. This is the only user that is currently can log into cloude Lucid id account and sync some basic info like Name, Profile picture etc.
  - **Secondary Driver** Any user added to the device other than the system user. Secondary users can be removed Primary Driver and cannot impact other users on a device. These users can run in the background and continue to have network connectivity.
  - **Guest Driver** Is also a secondary user created by default, Guest users have limited access to the features & control over the automobile

10.3 User Profile execution in Android 10:

10.3.1 two terms

- **UNLOCKS** : User profile state where in the User started in foreground and unlocked User account ready for user data access.
- **Suspend to RAM (S2R/STR)** : The standby Device stores the current operating status and other data in the memory, turns off the hard disk, peripherals and other equipment, and enters the waiting state. At this time, the memory still needs power to maintain its data, but the whole machine consumes very little power. When recovering, the computer reads data from the memory and returns to the state before the suspension, and the recovery speed is faster. Other names: Suspend, STR/S2R (Suspend To RAM), Suspend, Suspend to memory

10.3.2 A. From Very first boot

- CCC boots up and starts System user as headless user in background and Start and **UNLOCKS** the Primary Driver profile in foreground in PS:SLEEP.
- Displays are turned ON in PR:WINK
- On the even of Car turn off it pushes into PS:WINK and Display are turned OFF.

- **Suspend to RAM(S2R/STR)** trigged in PS:SLEEP and takes the snapshot of System User & Primary User.
- Followed by Power shutdown.

10.3.3 B. After Power Down.

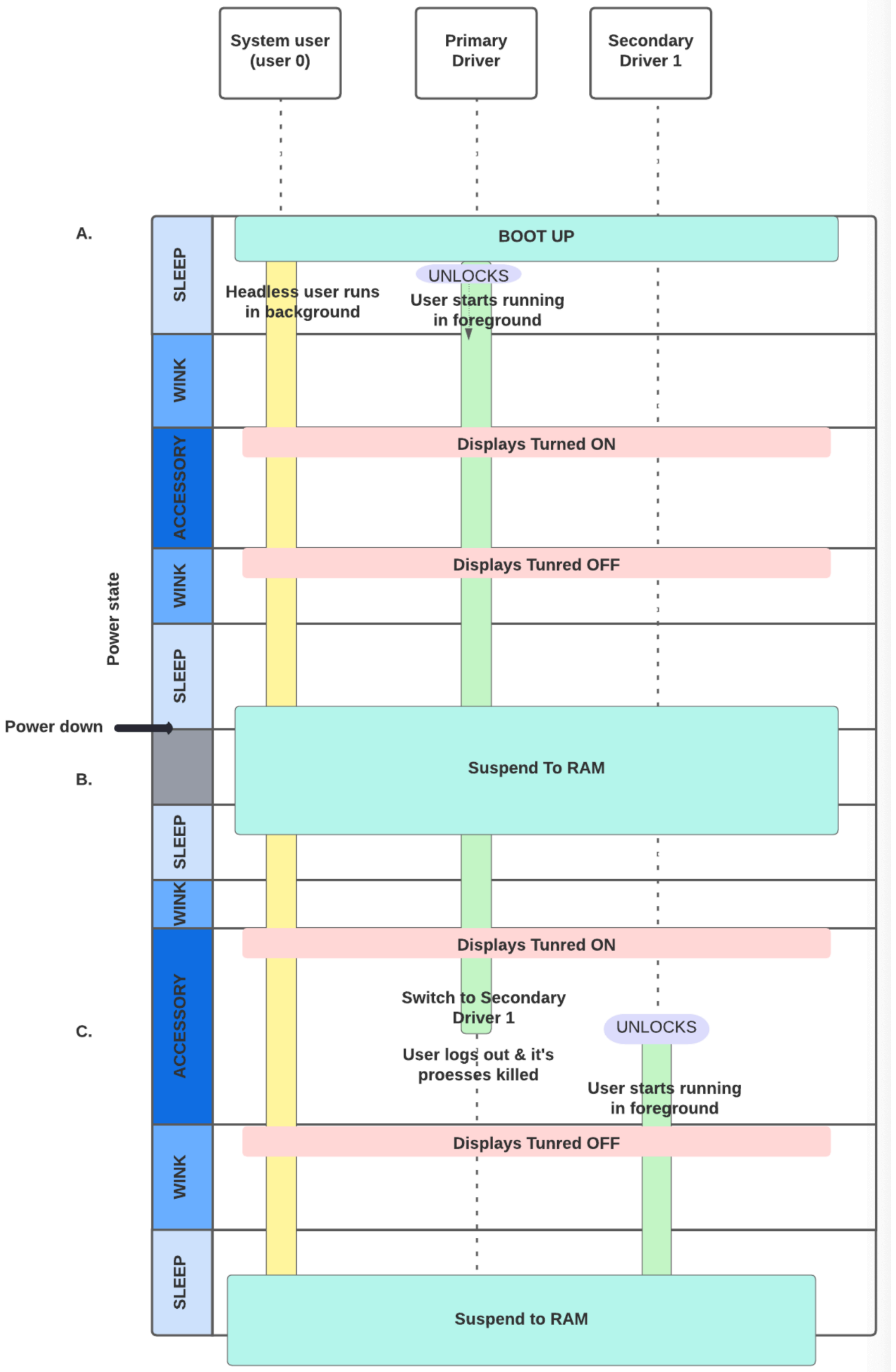
- Device resume from the RAM with the snap shot saved before Suspend in PS:SLEEP, as per the snapshot it will resumes the Headless user in background and Primary Driver in foreground.
- Turns on the Display in PS:WINK

10.3.4 C. Use case of Profile switch

- Create/Switched to Secondary Driver 1 from Primary driver in PS:ACCESSORY
- Unlocks the Secondary Driver 1 and runs user in foreground. PS: ACCESSORY
- Logs out the Primary Driver and kill all the related applications and processes PS:ACCESSORY
- When the car turned off it pushed power state to PS:WINK and turns of displays
- S2R is triggered in PS:SLEEP and takes the snap shot of Headless user and Secondary Driver 1 profile. and resumes from that snapshot for next power cycle.

10.3.5 flow chart shows above processes

- Following flowchart shows the User profile life cycle with respect to Vehicle Power state and Suspend to RAM* and other details:


System user (user 0)
Primary Driver
Secondary Driver 1
A.
BOOT UP
SLEEP
UNLOCKS
Headless user runs in background
User starts running in foreground
WINK
Displays Turned ON
ACCESSORY
Displays Tunred OFF
WINK
Power state
SLEEP
Power down
Suspend To RAM
B.
SLEEP
WINK
Displays Tunred ON
ACCESSORY
Switch to Secondary Driver 1
C.
UNLOCKS
User logs out & it's proesses killed
User starts running in foreground
Displays Tunred OFF
WINK
SLEEP
Suspend to RAM

11 LucidSystemUI APP Proguard related

```
 -include ../../libs/lucid-resource-lib/proguard/shrink.pro
 -ignorewarnings
 -keep public class com.lucid.car.sysui.ProfileManagerService {*;}
 -keep public class com.lucid.car.sysui.PinToDriveService {*;}
 -keep public class com.lucid.car.sysui.LucidSysUiService {*;}
```

- ../../libs/lucid-resource-lib/proguard/shrink.pro

```
-include android.pro
-include lucid-common.pro
-dontobfuscate
-dontoptimize
```

- lucid-common.pro

```
# keep standard entry points
-keep public class * extends android.app.Application
-keep public class * extends android.app.Service
-keep public class * extends android.content.BroadcastReceiver
-keep public class * extends android.content.ContentProvider
# keep ProtoBuf related classes
-keep class * extends com.google.protobuf.GeneratedMessageLite { *; }
-ignorewarnings
-dontwarn **
-dontwarn com.google.gson.reflect.TypeToken
```

- android.pro

```
-injars  bin/classes
-injars  libs
-outjars bin/classes-processed.jar
-libraryjars /usr/local/android-sdk/platforms/android-9/android.jar
#-libraryjars /usr/local/android-sdk/add-ons/google_apis-7_r01/libs/maps.jar

# ...
# Save the obfuscation mapping to a file, so you can de-obfuscate any stack
# traces later on.
-printmapping bin/classes-processed.map

# You can print out the seeds that are matching the keep options below.
#-printseeds bin/classes-processed.seeds
# Preverification is irrelevant for the dex compiler and the Dalvik VM.
-dontpreverify

# Reduce the size of the output some more.
-repackageclasses ''
-allowaccessmodification

# Switch off some optimizations that trip older versions of the Dalvik VM.
-optimizations !code/simplification/arithmetic

# Keep a fixed source file attribute and all line number tables to get line
# numbers in the stack traces.
# You can comment this out if you're not interested in stack traces.'
-renamesourcefileattribute SourceFile
-keepattributes SourceFile,LineNumberTable

# RemoteViews might need annotations.
-keepattributes *Annotation*

# Preserve all fundamental application classes.
-keep public class * extends android.app.Activity
-keep public class * extends android.app.Application
-keep public class * extends android.app.Service
-keep public class * extends android.content.BroadcastReceiver
-keep public class * extends android.content.ContentProvider

# Preserve all View implementations, their special context constructors, and
# their setters.
-keep public class * extends android.view.View {
    public <init>(android.content.Context);
    public <init>(android.content.Context, android.util.AttributeSet);
    public <init>(android.content.Context, android.util.AttributeSet, int);
    public void set*(...);
}

# Preserve all classes that have special context constructors, and the
# constructors themselves.
-keepclasseswithmembers class * {
    public <init>(android.content.Context, android.util.AttributeSet);
}

# Preserve all classes that have special context constructors, and the
# constructors themselves.
-keepclasseswithmembers class * {
    public <init>(android.content.Context, android.util.AttributeSet, int);
}

# Preserve all possible onClick handlers.
-keepclassmembers class * extends android.content.Context {
    public void *(android.view.View);
    public void *(android.view.MenuItem);
}

# Preserve the special fields of all Parcelable implementations.
-keepclassmembers class * implements android.os.Parcelable {
    static android.os.Parcelable$Creator CREATOR;
}

# Preserve static fields of inner classes of R classes that might be accessed
# through introspection.
-keepclassmembers class **.R$* {
    public static <fields>;
}

# Preserve annotated Javascript interface methods.
-keepclassmembers class * {
    @android.webkit.JavascriptInterface <methods>;
}

# Preserve the required interface from the License Verification Library
# (but don't nag the developer if the library is not used at all').
-keep public interface com.android.vending.licensing.ILicensingService
-dontnote com.android.vending.licensing.ILicensingService

# The Android Compatibility library references some classes that may not be
# present in all versions of the API, but we know that's ok.'
-dontwarn android.support.**

# Preserve all native method names and the names of their classes.
-keepclasseswithmembernames,includedescriptorclasses class * {
    native <methods>;
}

# Preserve the special static methods that are required in all enumeration
# classes.
-keepclassmembers,allowoptimization enum * {
    public static **[] values();
    public static ** valueOf(java.lang.String);
}

# Explicitly preserve all serialization members. The Serializable interface
# is only a marker interface, so it wouldn't save them.'
# You can comment this out if your application doesn't use serialization. '
# If your code contains serializable classes that have to be backward
# compatible, please refer to the manual.
-keepclassmembers class * implements java.io.Serializable {
    static final long serialVersionUID;
    static final java.io.ObjectStreamField[] serialPersistentFields;
    private void writeObject(java.io.ObjectOutputStream);
    private void readObject(java.io.ObjectInputStream);
    java.lang.Object writeReplace();
    java.lang.Object readResolve();
}
```