Desay API Description

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | **Date** | **Document Owner** | **Changes** |
| 1.0 | 2-28 | PanJuan | 初稿audio, MIC, CarProperty |
| 1.1 | 03-04 | PanJuan | 修改导航播报获取焦点的STREAM类型 |
| 1.2 | 03-06 | XieJun | 新增获取车身信息，发送多媒体信息等接口。 |
| 1.3 | 03-12 | XieJun | 新增车身信息回调接口，图像EVS接口等。 |
| 1.4 | 03-27 | XieJun | 调整接口，更新文档 |
| 1.5 | 04-10 | XieJun | 增加BaseManager，增加AudioManager接口，增加enhanceMemory接口 |
| 1.6 | 04-17 | XieJun | 1.增加获取AudioFocus信息接口；  2.增加蓝牙接口；  3.增加EV接口；  4.EVS增加doneWithFrame接口 |
| 1.7 | 05-21 | Qijun | 1.增加ev接口 |
| 1.7 | 05-21 | XieJun | 1.新增跳转界面说明 |
| 1.8 | 05-25 | QIJUN | 1.新增ev接口 2.新增em接口 3.新增 ea接口 |
| 1.9 | 05-28 | QIJUN | 1. 新增获取前一个源 2. 增加白天黑夜模式 3. 在线收音机的AudioSourceType： |
| 2.0 | 6-5 | QIJUN | 1,新增物理按键接口  2.新增空调aar接口  3.新增adas数据的接口  4.更新ev的接口  5.更新cedte的接口 |

目录

[1 android.media.AudioManager.java 8](#_Toc42508301)

[1.1 requestAudioFocus(AudioFocusRequest focusRequest) 8](#_Toc42508302)

[1.2 abandonAudioFocusRequest(AudioFocusRequest focusRequest） 8](#_Toc42508303)

[1.3 setStreamVolume (int streamType, int index, int flags) 9](#_Toc42508304)

[1.4 List<AudioFocusInfo> getAudioFocusInfo(int type)获取所有的音源list： 9](#_Toc42508305)

[1.5 String getCarAudioType(AudioFocusInfo info) 9](#_Toc42508306)

[1.6 在线收音机的AudioSourceType： 9](#_Toc42508307)

[2 Mic抢占 9](#_Toc42508308)

[3 平台界面跳转 10](#_Toc42508309)

[4 com.ford.base.FordBaseManager.java 10](#_Toc42508310)

[5 com.ford.car.FordCarInfoManager.java 10](#_Toc42508311)

[5.1 public static FordCarInfoManager getInstance(Context context) 10](#_Toc42508312)

[5.2 int getCarModel() 11](#_Toc42508313)

[5.3 String getVinCode() 11](#_Toc42508314)

[5.4 String getCarColor() 11](#_Toc42508315)

[5.5 String getESN() 11](#_Toc42508316)

[5.6 String getPCBANumber() 12](#_Toc42508317)

[5.7 String getAssemblyNumber() 12](#_Toc42508318)

[5.8 String getVMCUVersion() 12](#_Toc42508319)

[5.9 String getSerialNo() 12](#_Toc42508320)

[5.10 String getSystemModel() 12](#_Toc42508321)

[5.11 String getSystemVersion() 12](#_Toc42508322)

[5.12 int getCarPowerType() 12](#_Toc42508323)

[5.13 void setActiveStatus(int status) 12](#_Toc42508324)

[5.14 int getActiveStatus() 12](#_Toc42508325)

[5.15 String getSystemType() 13](#_Toc42508326)

[5.16 String getCarType() 13](#_Toc42508327)

[5.17 String getCarStyle() 13](#_Toc42508328)

[5.18 boolean has360Camera() 13](#_Toc42508329)

[6 com.ford.car.FordCarVendorExtensionManager.java 13](#_Toc42508330)

[6.1 void startServiceAsUser(Intent intent) 13](#_Toc42508331)

[6.2 String getBootAnimIsStopped() 13](#_Toc42508332)

[6.3 void setNeedLoginCheck(String loginCheck) 13](#_Toc42508333)

[6.4 String getNeedLoginCheck() 14](#_Toc42508334)

[6.5 void setAuthResult(int authed) 14](#_Toc42508335)

[6.6 String getVolumeState(String mountPoint) ； 14](#_Toc42508336)

[USB得挂载情况 14](#_Toc42508337)

[参数 14](#_Toc42508338)

[返回：unknown 没有挂载上 mounted usb已挂载 14](#_Toc42508339)

[6.7 int getDayAndNightMode() 14](#_Toc42508340)

[获取白天黑夜模式; 14](#_Toc42508341)

[6.8 void registerDayAndNightModeListener(ISvDataBusDispatcher.Stub listener) 14](#_Toc42508342)

[6.9 void unregisterDayAndNightModeListener(ISvDataBusDispatcher.Stub listener) 14](#_Toc42508343)

[6.10 int getPowerState() 15](#_Toc42508344)

[7 com.ford.car.FordCarSensorManager.java 15](#_Toc42508345)

[7.1 public static FordCarSensorManager getInstance(Context context) 15](#_Toc42508346)

[7.2 int getCarSpeedUnit() 15](#_Toc42508347)

[7.3 float getCarSpeed() 15](#_Toc42508348)

[7.4 int getCarDriveMode() 15](#_Toc42508349)

[7.5 int getCarDoorOpenStatus(int doorPos) 15](#_Toc42508350)

[7.6 float getCarEnduranceMileage() 16](#_Toc42508351)

[7.7 float getCarEndurancePercent() 16](#_Toc42508352)

[7.8 int getCarEnduranceWarnStatus() 16](#_Toc42508353)

[7.9 float getCarFuelPercent() 16](#_Toc42508354)

[7.10 int getCarFuelWarnStatus() 16](#_Toc42508355)

[7.11 int getCarGear() 17](#_Toc42508356)

[7.12 int getCarMileageUnit() 17](#_Toc42508357)

[7.13 float getCarTirePressure(int tirePos) 17](#_Toc42508358)

[7.14 int getCarTirePressureUnit() 17](#_Toc42508359)

[7.15 float getCarTireTemperature(int tirePos) 17](#_Toc42508360)

[7.16 int getCarTireTemperatureUnit() 18](#_Toc42508361)

[7.17 int getCarTireStatus(int tirePos) 18](#_Toc42508362)

[7.18 int getCarTireSupportResult() 18](#_Toc42508363)

[7.19 float getCarTotalMileage() 19](#_Toc42508364)

[7.20 int getCarWindowOpenStatus(int windowPos) 19](#_Toc42508365)

[7.21 addListener(SensorChangeListener listener) 19](#_Toc42508366)

[7.22 removeListener(SensorChangeListener listener) 19](#_Toc42508367)

[7.23 clearListeners() 19](#_Toc42508368)

[7.24 float getOutCarTemperature() 获取车外温度 返回值：温度的值 20](#_Toc42508369)

[7.25 回调接口 20](#_Toc42508370)

[7.25.1 驾驶模式 20](#_Toc42508371)

[7.25.2 车速 20](#_Toc42508372)

[7.25.3 点火状态 21](#_Toc42508373)

[7.25.4 里程 21](#_Toc42508374)

[7.25.5 里程单位 21](#_Toc42508375)

[7.25.6 油量 21](#_Toc42508376)

[7.25.7 档位 21](#_Toc42508377)

[7.25.8 车窗 22](#_Toc42508378)

[7.25.9 车门 22](#_Toc42508379)

[7.25.10 胎压 22](#_Toc42508380)

[7.26 sendNaviToTCU(Bundle bundle) 22](#_Toc42508381)

[8 com.ford.car.FordCarHvacManager.java 23](#_Toc42508382)

[8.1 public static FordCarHvacManager getInstance(Context context) 23](#_Toc42508383)

[8.2 boolean isCarFrontClimateOpen() 23](#_Toc42508384)

[8.3 boolean isCarRearClimateOpen() 23](#_Toc42508385)

[8.4 boolean isSupportAAR() 24](#_Toc42508386)

[8.5 int getCarClimateCycleMode() 24](#_Toc42508387)

[8.6 void setCarClimateCycleMode(int mode) 24](#_Toc42508388)

[8.7 void setCarFrontClimateOnOff(boolean isOpen) 24](#_Toc42508389)

[8.8 void setCarRearClimateOnOff(boolean isOpen) 24](#_Toc42508390)

[8.9 addListener(ClimateChangeListener listener) 24](#_Toc42508391)

[8.10 removeListener(ClimateChangeListener listener) 25](#_Toc42508392)

[8.11 clearListeners() 26](#_Toc42508393)

[9 com.ford.media.MediaEventManager 26](#_Toc42508394)

[9.1 USB广播 26](#_Toc42508395)

[9.2 public static FordMediaEventManager getInstance(Context context) 26](#_Toc42508396)

[9.3 void sendVRTips(String text) 26](#_Toc42508397)

[9.4 void sendID3Info(ID3Info id3Info) 27](#_Toc42508398)

[9.5 void sendPlayState(int state) 27](#_Toc42508399)

[9.6 void sendPlayMode(int mode) 27](#_Toc42508400)

[9.7 addListener(ActionListener listener) 27](#_Toc42508401)

[9.8 removeListener(ActionListener listener) 27](#_Toc42508402)

[9.9 clearListeners() 27](#_Toc42508403)

[9.10 回调接口 28](#_Toc42508404)

[10 com.ford.camera.EVSCameraManager 28](#_Toc42508405)

[11 com.ford.vendor.FordEnhanceMemoryManager 28](#_Toc42508406)

[11.1 void changeDriverProfile(int index) 28](#_Toc42508407)

[11.2 interface IOnProfileActivebyHmi{ 28](#_Toc42508408)

[11.3 interface IOnProfileActivebyKey{ 29](#_Toc42508409)

[11.4 boolean getGlobalProperty(int propId); 29](#_Toc42508410)

[11.5 void createDriverProfile(int index); 29](#_Toc42508411)

[11.6 interface IOnProfileCreate{ 29](#_Toc42508412)

[11.7 interface IPaakConnectionChanged{ 30](#_Toc42508413)

[11.8 void requestPhonePairing(int profileIndex); 30](#_Toc42508414)

[11.9 void requestPhoneUnPairing(int profileIndex); 30](#_Toc42508415)

[11.10 void requestPhoneOverwritePairing(int profileIndex); 30](#_Toc42508416)

[11.11 interface PhoneAssoProfileCallback{ 31](#_Toc42508417)

[11.12 void requestCarKeyPairing(int profileIndex); 31](#_Toc42508418)

[11.13 void requestCarKeyUnPairing(int profileIndex); 31](#_Toc42508419)

[11.14 void requestCarKeyOverwritePairing(int profileIndex); 32](#_Toc42508420)

[11.15 interface CarKeyAssoProfileCallback{ 32](#_Toc42508421)

[11.16 byte[] getProfilesKeyFobAssociated(); 33](#_Toc42508422)

[11.17 byte[] getProfilesKeyPhoneAssociated(); 33](#_Toc42508423)

[11.18 interface IOnProfilesDelete{ 33](#_Toc42508424)

[11.19 void requestDeleteProfiles(int[] indexes) 33](#_Toc42508425)

[11.20 void stopCarKeyPairing(int profileIndex); 34](#_Toc42508426)

[11.21 void stopPhonePairing(int profileIndex); 34](#_Toc42508427)

[11.22 interface IOnGearChanged{ 34](#_Toc42508428)

[11.23 int getGear(); 34](#_Toc42508429)

[11.24 List<String> getCollectRadios() 35](#_Toc42508430)

[11.25 void setCollectRadios(List<String> radios) 35](#_Toc42508431)

[11.26 interface ICollectedRadiosChange{ 35](#_Toc42508432)

[11.27 List<String> getAutoSavedRadios() 自动存储电台的读取，最多18个电台 35](#_Toc42508433)

[11.28 void setAutoSavedRadios(List<String> radios) 35](#_Toc42508434)

[11.29 interface IAutoSavedRadiosChange{ 35](#_Toc42508435)

[11.30 int getTheme() 主题的读取 36](#_Toc42508436)

[11.31 void setTheme(int theme) 主题的设置 36](#_Toc42508437)

[11.32 interface IThemeChange{ 36](#_Toc42508438)

[11.33 int getDistanceUnit() 获取距离单位 36](#_Toc42508439)

[11.34 void setDistanceUnit(int unit) 设置距离单位 36](#_Toc42508440)

[11.35 interface IDistanceUnitChange{ 36](#_Toc42508441)

[11.36 int getTemperatureUnit() 获取温度单位 37](#_Toc42508442)

[11.37 void setTemperatureUnit(int unit) 设置温度单位 37](#_Toc42508443)

[11.38 interface ITemperatureUnitChange{ 37](#_Toc42508444)

[11.39 int getTyreUnit() 获取胎压单位，默认值kPa 37](#_Toc42508445)

[11.40 void setTyreUnit(int tyreUnit) 设置胎压单位 37](#_Toc42508446)

[11.41 interface ITyreUnitChange{ 37](#_Toc42508447)

[11.42 int getMeasureUnit() 获取测量单位，默认值是km&km/l 38](#_Toc42508448)

[11.43 void setMeasureUnit(int measureUnit ) 设置测量单位 38](#_Toc42508449)

[11.44 interface IMeasureUnitChange{ 38](#_Toc42508450)

[11.45 int getAmbientLampSwitch() 获取氛围灯开关 38](#_Toc42508451)

[11.46 void setAmbientLampSwitch(int switch) 设置氛围灯开关 38](#_Toc42508452)

[11.47 interface IAmbientLampSwitchChange{ 38](#_Toc42508453)

[11.48 int getAmbientColor() 获取氛围灯颜色 39](#_Toc42508454)

[11.49 void setAmbientColor(int color ) 设置氛围灯颜色 39](#_Toc42508455)

[11.50 interface IAmbientColorChange{ 39](#_Toc42508456)

[11.51 10.51 int getAmbientBrightness() 获取氛围灯亮度 39](#_Toc42508457)

[11.52 void setAmbientBrightness(int brightness) 设置氛围灯亮度 39](#_Toc42508458)

[11.53 interface IAmbientBrightnessChange{ 39](#_Toc42508459)

[11.54 boolean getReverseImageDelay() 获取倒车影像延迟 40](#_Toc42508460)

[11.55 void setReverseImageDelay(boolean delay) 设置倒车影像延迟 40](#_Toc42508461)

[11.56 interface IReverseImageDelayChange{ 40](#_Toc42508462)

[11.57 int getSpeedVolCompensation() 获取车速音量补偿设置 40](#_Toc42508463)

[11.58 void setSpeedVolCompensation(int compensation) 设置车速音量补偿设置 40](#_Toc42508464)

[11.59 interface ISpeedVolCompensationChange{ 40](#_Toc42508465)

[11.60 int getAirConTemperature() 获取空调温度 41](#_Toc42508466)

[11.61 void setAirConTemperature(int temperature) 设置空调温度 41](#_Toc42508467)

[11.62 interface IAirConTemperatureChange{ 41](#_Toc42508468)

[11.63 void setDriveMode(int mode) 设置驾驶模式 41](#_Toc42508469)

[12 com.ford.vendor.FordEVChargeManager 41](#_Toc42508470)

[12.1 byte getPlugConnectStatus(); 41](#_Toc42508471)

[12.2 interface IPlugConnectStatus{ 41](#_Toc42508472)

[12.3 boolean isChargePlanPos(); 42](#_Toc42508473)

[12.4 IVehiclePosChange { 42](#_Toc42508474)

[12.5 boolean getChargPlanSwitch(); 42](#_Toc42508475)

[12.6 IChargPlanSwitchChange{ 42](#_Toc42508476)

[12.7 void setChargPlanSwitch(boolean on); 42](#_Toc42508477)

[12.8 byte getChargePercentage(); 43](#_Toc42508478)

[12.9 interface IChargeProcessChange{ 43](#_Toc42508479)

[12.10 ChargePlan getCurentChargePlan(); 43](#_Toc42508480)

[12.11 byte getChrgStat(); 43](#_Toc42508481)

[12.12 interface IChrgStatChange{ 43](#_Toc42508482)

[12.13 int getRemainingMileage(); 44](#_Toc42508483)

[12.14 float getfullyChargeHours(); 44](#_Toc42508484)

[12.15 byte getChargeTargetPercent(); 44](#_Toc42508485)

[12.16 ChargeStartTime getChargeStartTime(); 44](#_Toc42508486)

[12.17 ChargeEndTime getChargeEndTime(); 44](#_Toc42508487)

[12.18 DepartAndComfort getDepartAndComfort(); 45](#_Toc42508488)

[12.19 boolean isVehicleRun(); 45](#_Toc42508489)

[12.20 interface IVehicleRunChange{ 45](#_Toc42508490)

[12.21 List<ChargeLocation> getUnsavedChargeLocations(); 45](#_Toc42508491)

[12.22 List<ChargePlan> getChargePlans(); 45](#_Toc42508492)

[12.23 void insertChargePlan(ChargePlan chargePlan); 46](#_Toc42508493)

[12.24 void updateChargePlan(ChargePlan chargePlan); 46](#_Toc42508494)

[12.25 void deleteChargePlan(int id); 46](#_Toc42508495)

[12.26 Interface IEditChargePlan{ 46](#_Toc42508496)

[12.27 List<DepartAndComfort> getDepartAndComforts(); 46](#_Toc42508497)

[12.28 void setDepartAndComfortSwitch(boolean on); 47](#_Toc42508498)

[12.29 IDepartAndComfortSwitchChange{ 47](#_Toc42508499)

[12.30 void saveDepartAndComforts(List<DepartAndComfort> departAndComforts); 47](#_Toc42508500)

[12.31 void deleteDepartAndComforts(List<byte> ids); 47](#_Toc42508501)

[12.32 void deleteAllDepartAndComforts(); 47](#_Toc42508502)

[12.33 IEditDepartAndComfort { 47](#_Toc42508503)

[12.34 interface IChargeAndDepartConflict{ 48](#_Toc42508504)

[12.35 void disConnection() 取消相关service的连接 48](#_Toc42508505)

[12.36 interface IRemainMileageListener{ 48](#_Toc42508506)

[12.37 byte getBattLowWarning(); 48](#_Toc42508507)

[12.38 interface IBattLowWarning{ 48](#_Toc42508508)

[12.39 byte getChrgrInPwType() 49](#_Toc42508509)

[12.40 interface IChrgrInPwTypeChange 49](#_Toc42508510)

[12.41 interface IRemainingMileageChange 49](#_Toc42508511)

[12.42 interface IChargeEndTimeChange 49](#_Toc42508512)

[13 com.ford.vendor.FordBluetoothManager 50](#_Toc42508513)

[14 com.ford.vendor.FordCeDTEManager 50](#_Toc42508514)

[14.1 void setTripInfoStructure(byte[] tripInfoStructure) 50](#_Toc42508515)

[14.2 void setTripStatus(byte[] tripStatus) 51](#_Toc42508516)

[14.3 void setConnectorType(byte[] connectorType) 51](#_Toc42508517)

[14.4 void setBatteryPackType(byte[] batteryPackType) 51](#_Toc42508518)

[14.5 void onConnect(FordBaseManager.ConnectionCallback connection) 连接服务，连接后同步返回数据后才有信号数据返回 51](#_Toc42508519)

[14.6 void onDisConnect() 断开服务连接 51](#_Toc42508520)

[14.7 int getRemainingMileage() 51](#_Toc42508521)

[15 com.ford.ea.FordEmergencyManager 51](#_Toc42508522)

[15.1 void sendReqWithRsp(BaseReqMessage baseReqMessage) 51](#_Toc42508523)

[15.2 int getCrashCan() 51](#_Toc42508524)

[15.3 int getIgnKeyType\_D\_Actl() 52](#_Toc42508525)

[*15.4* *int getMyKey\_e911Override\_St()* 52](#_Toc42508526)

[15.5 int getDrStatDrv\_B\_Actl() 52](#_Toc42508527)

[15.6 int getIgnition\_Status() 52](#_Toc42508528)

[15.7 int getPOWER\_MODE() 52](#_Toc42508529)

[15.8 int getVEDS() 52](#_Toc42508530)

[15.9 public interface IRspCallback { 52](#_Toc42508531)

[16 com.ford.vendor.FordBtManager 52](#_Toc42508532)

[16.1 void connect(String address) 52](#_Toc42508533)

[16.2 void dial(String address, String number) 53](#_Toc42508534)

[16.3 void terminateCall(String address) 53](#_Toc42508535)

[16.4 void connectAudio(String address) 53](#_Toc42508536)

[16.5 void disconnectAudio(String address) 53](#_Toc42508537)

[16.6 interface IBtHfpChangeListener { 53](#_Toc42508538)

[17 com.ford.audio. FordAudioManager 54](#_Toc42508539)

[17.1 public static final String CAR\_AUDIO\_TYPE\_EA= 54](#_Toc42508540)

[18 com.ford.audio.FordCommonManager 54](#_Toc42508541)

[18.1 void connect(ConnectionCallback connection) 启动服务，并同步将消息发送出去 54](#_Toc42508542)

[18.2 void disConnect() 54](#_Toc42508543)

[18.3 sendReqWithRsp(BaseReqMessage baseReqMessage) 54](#_Toc42508544)

[18.4 sendRequestNoRsp(BaseReqMessage baseReqMessage) 发送不带返回消息的接口 BaseReqMessage 救援相关的参数 55](#_Toc42508545)

[18.5 void setResponseListener(IResponseCallback responseListener) 55](#_Toc42508546)

[18.6 interface IResponseCallback { 55](#_Toc42508547)

[19 com.ford.vendor.FordHardKeyManager 55](#_Toc42508548)

[19.1 void registerKeyEvent(int AppID, IkeyEventListener listener); 根据相应的AppID注册物理按键 55](#_Toc42508549)

[19.2 void unregisterKeyEvent(int AppID, IkeyEventListener listener) 56](#_Toc42508550)

[19.3 interface IkeyEventListener 56](#_Toc42508551)

[20 com.ford.car.FordCarClimmateManager 60](#_Toc42508552)

[20.1 int getGlobalProperty(int type) 60](#_Toc42508553)

[20.2 interface AARChangeListener 60](#_Toc42508554)

[20.3 int getGlobalProperty1(int type) 61](#_Toc42508555)

[20.4 void setGlobalProperty(int type, int value) 61](#_Toc42508556)

[20.5 void setNewAirRefresh() 62](#_Toc42508557)

[20.6 int getNewAirRefresh() 62](#_Toc42508558)

[20.7 int getAcConfig() 62](#_Toc42508559)

[21 com.ford.vendor.FordElectronicHorizonManager 62](#_Toc42508560)

[21.1 void onConnect(FordBaseManager.ConnectionCallback connection) 62](#_Toc42508561)

[21.2 void onDisConnect() 62](#_Toc42508562)

[21.3 ElectronicHorizon1\_Rsp getElectronicHorizon1\_Rsp() 63](#_Toc42508563)

[21.4 ElectronicHorizon2\_Rsp getElectronicHorizon2() 63](#_Toc42508564)

# android.media.AudioManager.java

## requestAudioFocus(AudioFocusRequest focusRequest)

Android原生接口，用于申请焦点，需要构造一个audioAttributes，source类型通过audioAttributes的usage传递给系统。

具体实现可以遵循原生的接口使用：https://developer.android.google.cn/reference/android/media/AudioManager?hl=en#requestAudioFocus(android.media.AudioFocusRequest)

其中，导航播报的参数如下：

CONTENT\_TYPE: com.ford.audio.FordAudioManager.STREAM\_NAVIGATION；

正常播报USAGE:  AudioAttributes.USAGE\_ASSISTANCE\_NAVIGATION\_GUIDANCE；

导航复播USAGE: AudioAttributes.USAGE\_ASSISTANCE\_ACCESSIBILITY。

注：导航复播时，焦点优先级最高，比语音都高

## abandonAudioFocusRequest(AudioFocusRequest focusRequest）

abandonAudioFocus的audioAttributes参数需要和其requestAudioFocus时传入的attribute对应  
**特别重要：**request时的attribute和abandon时的attribute必须为同一个attribute。

## setStreamVolume (int streamType, int index, int flags)

Android原生接口，调节音量。

其中，导航音量传入的参数streamtype对应com.ford.audio.FordAudioManager.STREAM\_NAVIGATION。

## List<AudioFocusInfo> getAudioFocusInfo(int type)获取所有的音源list：

参数type：传AUDIOFOCUS\_ALL\_TYPE 即 0

返回：所有的音源list

## String getCarAudioType(AudioFocusInfo info)

1.获取当前源 2.获取上一个源  
实现过程：

获取所有得音源AudioFocusInfo得infoList,即根据1.4的接口获取音源list;

infoList.get(0)获取当前源得AudioFocusInfo，

infoList.get(1)获取上一个源AudioFocusInfo

根据AudioFocusInfo获取这个源得类型car\_audio\_type

return null:没有当前源或者没有前一个源 notnull:源得类型

## 在线收音机的AudioSourceType：

public static final String CAR\_AUDIO\_TYPE\_ONLINE\_NETWORKSTATION = SvCarAudioManager.CAR\_AUDIO\_TYPE\_ONLINE\_NETWORKSTATION;

# Mic抢占

1. VR APP自己定义一个public static final int HOTWORD = 1999;
2. Manifest.xml加上权限”android.permission.CAPTURE\_AUDIO\_HOTWORD”
3. 语音app在收到其他apps抢占mic的消息后，需要将降噪设置设回正常模式，以免影响其他app的mic使用。

# 平台界面跳转

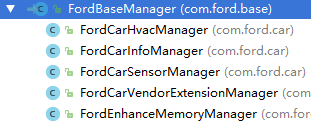
intent.setClassName("com.desay.setting", "com.desay.settings.systemSet.activity.WifiActivity"); //进入WiFi设置界面

intent.setClassName("com.desay.setting", "com.desay.settings.systemSet.bluetooth.activity.BluetoothActivity"); // 进入蓝牙设置界面

intent.setClassName("com.desay.setting", "com.desay.settings.activity.MainActivity"); // 进入车辆控制界面

# com.ford.base.FordBaseManager.java

BaseManager为各个Manager基类，提供connect和disconnect接口。



继承自BaseManager的管理类，调用方式都是xxxxManager.getInstance().connect(ServiceConnection)，收到onServiceConnected回调后，表示Service连接成功，可以开始业务逻辑。

# com.ford.car.FordCarInfoManager.java

## public static FordCarInfoManager getInstance(Context context)

获取单例。

## int getCarModel()

获取当前车型。

返回值说明：

**public static final int *CAR\_MODEL\_CD542H*** = 0;*//CD542-High***public static final int *CAR\_MODEL\_CD542M*** = 1;*//CD542-Middle***public static final int *CAR\_MODEL\_CD542L*** = 2;*//CD542-Low***public static final int *CAR\_MODEL\_CX727H*** = 3;*//CX727-High***public static final int *CAR\_MODEL\_CX727L*** = 4;*//CX727-Low***public static final int *CAR\_MODEL\_U725H*** = 5;*//U725-High***public static final int *CAR\_MODEL\_U725L*** = 6;*//U725-Low***public static final int *CAR\_MODEL\_P702*** = 7;*//P702***public static final int *CAR\_MODEL\_U554H*** = 8;*//U554-High***public static final int *CAR\_MODEL\_U554L*** = 9;*//U554-Low***public static final int *CAR\_MODEL\_U625H*** = 10;*//U625-High***public static final int *CAR\_MODEL\_U625M*** = 11;*//U625-Middle***public static final int *CAR\_MODEL\_U625L*** = 12;*//U625-Low***public static final int *CAR\_MODEL\_CD764*** = 13;*//CD764*

## String getVinCode()

获取Vin码。

## String getCarColor()

获取车辆颜色

返回值：

**public static final** String ***CAR\_COLOR\_BLACK*** = **"black"**;  
**public static final** String ***CAR\_COLOR\_WHITE*** = **"white"**;  
**public static final** String ***CAR\_COLOR\_RED*** = **"red"**;  
**public static final** String ***CAR\_COLOR\_SILVER*** = **"silver"**;

## String getESN()

获取车机设备电子序列号。

## String getPCBANumber()

获取车机PCBA零件号。

## String getAssemblyNumber()

获取车机设备型号。

## String getVMCUVersion()

获取VMCU版本号。

## String getSerialNo()

获取序列号，等效于SystemProperties.*get*(**"ro.serialno"**);

## String getSystemModel()

获取系统信号，等效于SystemProperties.*get*(**"ro.product.model"**);

## String getSystemVersion()

获取系统版本，等效于SystemProperties.*get*(**"ro.build.id"**);

## int getCarPowerType()

获取车辆动力类型

返回值，0：油，1：电，2：BEV

**public static final int *POWER\_TYPE\_OIL*** = 0;//油  
**public static final int *POWER\_TYPE\_ELE*** = 1;//电  
**public static final int *POWER\_TYPE\_BEV*** = 2;//BEV

## void setActiveStatus(int status)

写激活状态。

## int getActiveStatus()

读取激活状态。

## String getSystemType()

获取ROM版本类型，debug版本返回“debug”，release版本返回“release”。

## String getCarType()

获取车辆样式，返回值范围：*"4DR", "SUV", "5DR"*

## String getCarStyle()

获取车辆款式，返回值范围：*"DI","DJ","DP","DF","DG","DM","DH","DL","DX","DR","MP","DQ","DN","DM","DJ","DK","GV","TP","DD","HZ","JY"*

## boolean has360Camera()

是否有360Camera。true表示有360，false表示没有。

# com.ford.car.FordCarVendorExtensionManager.java

## void startServiceAsUser(Intent intent)

对外提供startServiceAsUser接口：



## String getBootAnimIsStopped()

开机动画是否停止。返回“stopped”表示已停止。

## void setNeedLoginCheck(String loginCheck)

设置是否强制检验账号登录。"logincheckclose"表示免登录，"logincheckopen"表示需要强制登录。

## String getNeedLoginCheck()

获取是否强制检验账号登录。"logincheckclose"表示免登录，"logincheckopen"表示需要强制登录。

## void setAuthResult(int authed)

保存账号授权协议的结果。1：已授权，0：未授权。

## String getVolumeState(String mountPoint) ；

USB得挂载情况

参数**mountPoint**：**public static final String *STORAGE\_USB* = "/storage/usb";**

返回：unknown 没有挂载上 mounted usb已挂载

## int getDayAndNightMode()

## 获取白天黑夜模式;

返回; public static final int GET\_MODE\_FAIL = -1;失败

public static final int GET\_DAY\_MODE = 0;白天

public static final int GET\_NIGHT\_MODE = 1;黑夜

## void registerDayAndNightModeListener(ISvDataBusDispatcher.Stub listener)

注册白天黑夜模式的监听

## void unregisterDayAndNightModeListener(ISvDataBusDispatcher.Stub listener)

反注册白天黑夜模式的监听

## int getPowerState()

获取电源的模式

# com.ford.car.FordCarSensorManager.java

## public static FordCarSensorManager getInstance(Context context)

获取Manager单例。

## int getCarSpeedUnit()

获取车速单位。

返回值说明：

**public static final int *SPEED\_UNIT\_CMS*** = 0;*// "cm/s";***public static final int *SPEED\_UNIT\_MS*** = 1;*//"m/s";***public static final int *SPEED\_UNIT\_KMH*** = 2;*//"km/h";*

## float getCarSpeed()

获取车速。

## int getCarDriveMode()

获取驾驶模式

返回值说明：

**public static final int *DRIVE\_MODE\_SOLO*** = 0;*//solo mode***public static final int *DRIVE\_MODE\_CO\_PILOT*** = 1;*//co-pilot mode***public static final int *DRIVE\_MODE\_INDIVIDUAL*** = 2;*//individual mode*

## int getCarDoorOpenStatus(int doorPos)

获取各位置的车门打开状态。

参数：

**public static final int *DOOR\_FRONT\_LEFT*** = 0; //左前车门  
**public static final int *DOOR\_FRONT\_RIGHT*** = 1; //右前车门  
**public static final int *DOOR\_REAR\_LEFT*** = 2; //左后车门  
**public static final int *DOOR\_REAR\_RIGHT*** = 3; //右后车门  
**public static final int *DOOR\_OUTSIDE\_TRUNK*** = 4; //后备箱外车门  
**public static final int *DOOR\_INSIDE\_TRUNK*** = 5; //后备箱内车门

返回值，1：关闭，2：打开

**public static final int *WINDOW\_DOOR\_STATUS\_CLOSE*** = 1;  
**public static final int *WINDOW\_DOOR\_STATUS\_OPEN*** = 2;

## float getCarEnduranceMileage()

获取车辆续航里程

## float getCarEndurancePercent()

获取车辆续航百分比

## int getCarEnduranceWarnStatus()

获取续航警告状态

返回值，0：正常，1：告警

**public static final int *ENDURANCE\_STATUS\_OK*** = 0; **public static final int *ENDURANCE\_STATUS\_WARN*** = 1;

## float getCarFuelPercent()

获取剩余油量百分比。比如返回值为78.1，代表还剩余78.1%的油量。

## int getCarFuelWarnStatus()

获取油量告警状态

返回值，0：正常，1：告警

**public static final int *FUEL\_STATUS\_OK*** = 0; *//fuel status is ok***public static final int *FUEL\_STATUS\_WARN*** = 1; *//fuel status is warning*

## int getCarGear()

获取车辆档位

返回值，0：N档，1：P档，2：R档，3：D档

**public static final int *GEAR\_NEUTRAL*** = 0; *//N***public static final int *GEAR\_PARKING*** = 1; *//P***public static final int *GEAR\_REVERSE*** = 2; *//R***public static final int *GEAR\_DRIVE*** = 3; *//D*

## int getCarMileageUnit()

获取里程单位

返回值，

**public static final int *MILEAGE\_UNIT\_MILE*** = 0;*//"英里"***public static final int *MILEAGE\_UNIT\_KM*** = 1;*//"公里"*

## float getCarTirePressure(int tirePos)

获取胎压

参数：

**public static final int *TIRE\_FRONT\_LEFT*** = 0;//前左轮胎  
**public static final int *TIRE\_FRONT\_RIGHT*** = 1;//前右轮胎  
**public static final int *TIRE\_REAR\_LEFT*** = 2;//后左轮胎  
**public static final int *TIRE\_REAR\_RIGHT*** = 3;//后右轮胎

## int getCarTirePressureUnit()

获取胎压单位

返回值，

**public static final int *TIRE\_PRESSURE\_UNIT\_BAR*** = 0;*//bar***public static final int *TIRE\_PRESSURE\_UNIT\_KPA*** = 1;*//kpa*

## float getCarTireTemperature(int tirePos)

获取胎温

参数：

**public static final int *TIRE\_FRONT\_LEFT*** = 0;//前左轮胎  
**public static final int *TIRE\_FRONT\_RIGHT*** = 1;//前右轮胎  
**public static final int *TIRE\_REAR\_LEFT*** = 2;//后左轮胎  
**public static final int *TIRE\_REAR\_RIGHT*** = 3;//后右轮胎

## int getCarTireTemperatureUnit()

获取胎温单位

返回值，

**public static final int *TIRE\_TEMPERATURE\_UNIT\_CEN*** = 0;*//centigrade*

## int getCarTireStatus(int tirePos)

获取轮胎状态

参数：

**public static final int *TIRE\_FRONT\_LEFT*** = 0;//前左轮胎  
**public static final int *TIRE\_FRONT\_RIGHT*** = 1;//前右轮胎  
**public static final int *TIRE\_REAR\_LEFT*** = 2;//后左轮胎  
**public static final int *TIRE\_REAR\_RIGHT*** = 3;//后右轮胎

返回值：

**public static final int *TIRE\_STATUS\_NORMAL*** = 1; //正常状态  
**public static final int *TIRE\_STATUS\_UNNORMAL*** = 2; //异常状态

## int getCarTireSupportResult()

车辆是否支持获取胎温、胎压

返回值，

**public static final int *SUPPORT\_TIRE\_NO*** = 0; *//都不支持***public static final int *SUPPORT\_TIRE\_TEMP\_PRESSURE*** = 1; *//都支持***public static final int *SUPPORT\_TIRE\_PRESSURE*** = 2; *//仅支持获取胎压*

## float getCarTotalMileage()

获取车辆总里程

## int getCarWindowOpenStatus(int windowPos)

获取车窗打开状态

参数，

**public static final int *WINDOW\_POS\_DRIVER*** = 0; //驾驶位车窗  
**public static final int *WINDOW\_POS\_FRONT\_PASSENGER*** = 1; //副驾车窗  
**public static final int *WINDOW\_POS\_REAR\_DRIVER*** = 2; //驾驶位后排车窗  
**public static final int *WINDOW\_POS\_REAR\_PASSENGER*** = 3; //副驾位后排车窗

返回值，1：关闭，2：打开

**public static final int *WINDOW\_DOOR\_STATUS\_CLOSE*** = 1;  
**public static final int *WINDOW\_DOOR\_STATUS\_OPEN*** = 2;

## addListener(SensorChangeListener listener)

增加listener。

## removeListener(SensorChangeListener listener)

移除listener。

## clearListeners()

清空listener集合。

## float getOutCarTemperature() 获取车外温度 返回值：温度的值

## 回调接口

**public interface** SensorChangeListener {  
  
 **void** onSensorChange(**int** eventType, **int**[] intValues, **float**[] floatValues);  
}

FordCarSensorManager的各种传感器类信息，例如车速、档位等变化，统一回调onSensorChange接口。eventType表示对应事件类型，intValues保存回调的int型值，floatValues保存回调的float型值。

例如，收到onSensorChange(0, intValues, floatValues)，eventType为0，代表驾驶模式发生了变化，驾驶模式值是一个int型，所以取intValues[0]即可；没有float值，floatValues为null。

### 驾驶模式

eventType:**public static final int *EVENT\_TYPE\_DRIVE\_MODE*** = 0;

回调值：

intValues[0]，代表驾驶模式的值，值定义见4.4节。

### 车速

eventType:**public static final int *EVENT\_TYPE\_SPEED\_VALUE*** = 1;

回调值：

floatValues[0]，代表车速值

### 点火状态

eventType:**public static final int *EVENT\_TYPE\_IGNITION*** = 2;

回调值：

intValues[0]，代表点火状态值，值定义：

**public static final int *IGNITION\_STATUS\_OFF*** = 0;*//ignition-off***public static final int *IGNITION\_STATUS\_ON*** = 1;*//ignition-on*

### 里程

eventType:**public static final int *EVENT\_TYPE\_MILEAGE\_VALUE*** = 3;

回调值：

floatValues[0]，代表当前的续航里程；floatValues[1]，代表总里程。

### 里程单位

eventType:**public static final int *EVENT\_TYPE\_MILEAGE\_UNIT*** = 4;

回调值：

intValues[0]，代表里程单位，值定义见4.12节。

### 油量

eventType:**public static final int *EVENT\_TYPE\_FUEL*** = 5;

回调值：

intValues[0]，代表油量告警状态，值定义见4.11节。

floatValues[0]，代表剩余油量百分比。

### 档位

eventType:**public static final int *EVENT\_TYPE\_GEAR*** = 6;

回调值：

intValues[0]，代表档位，值定义见4.13节。

### 车窗

eventType:**public static final int *EVENT\_TYPE\_WINDOW*** = 7;

回调值：

intValues[0]，代表车窗位置，值定义见4.20节。

intValues[1]，代表车窗开关状态，值定义见4.20节。

### 车门

eventType:**public static final int *EVENT\_TYPE\_DOOR*** = 8;

回调值：

intValues[0]，代表车门位置，值定义见4.5节。

intValues[1]，代表车门开关状态，值定义见4.5节。

### 胎压

public static final int EVENT\_TYPE\_TIRE\_PRESURE\_VALUE = 9;

## sendNaviToTCU(Bundle bundle)

将处理后的导航信息，发送给车机系统。

Bundle参数，用法与原先保持一致：



# com.ford.car.FordCarHvacManager.java

## public static FordCarHvacManager getInstance(Context context)

获取Manager单例。

## boolean isCarFrontClimateOpen()

前排空调是否打开。

返回值，true：打开，false：关闭

## boolean isCarRearClimateOpen()

后排空调是否打开。

返回值，true：打开，false：关闭

## boolean isSupportAAR()

是否有AAR功能。

返回值，true：有，false：无

## int getCarClimateCycleMode()

获取空调循环模式。

返回值，0：内循环，1：外循环

**public static final int *CLIMATE\_CYCLE\_MODE\_INNER*** = 0;  
**public static final int *CLIMATE\_CYCLE\_MODE\_OUTER*** = 1;

## void setCarClimateCycleMode(int mode)

设置空调循环模式。

参数，0：内循环，1：外循环

**public static final int *CLIMATE\_CYCLE\_MODE\_INNER*** = 0;  
**public static final int *CLIMATE\_CYCLE\_MODE\_OUTER*** = 1;

## void setCarFrontClimateOnOff(boolean isOpen)

设置前排空调开关。

参数，true：打开空调，false：关闭空调

## void setCarRearClimateOnOff(boolean isOpen)

设置后排空调开关。

参数，true：打开空调，false：关闭空调

## addListener(ClimateChangeListener listener)

设置空调状态变化的监听器。

**public interface** ClimateChangeListener {  
  
 */\*\*  
 \* Called when the climate's open status has changed.  
 \** ***@param postion*** *which climate's open status has changed.  
 \* 0: the front climate  
 \* 1: the rear climate  
 \** ***@param isOpen*** *climate's open status value.  
 \* 0: open  
 \* 1: close  
 \** ***@param operation*** *How did the user turn the climate on/off  
 \* 0: use app on the screen, default value.  
 \* 1: use hardKey on the car.  
 \*/* **void** onOpenStatusChange(**int** postion, **boolean** isOpen, **int** operation);  
  
 */\*\*  
 \* Called when the climate's cycle mode has changed.  
 \** ***@param mode*** *value after mode change  
 \*/* **void** onCycleModeChange(**int** mode);  
  
 */\*\*  
 \* Called when the climate's change has error.  
 \** ***@param errorCode*** *error code  
 \* 100: the front climate open error  
 \* 101: the front climate close error  
 \* 110: the rear climate open error  
 \* 111: the rear climate close error  
 \* 200: the climate cycle mode change error  
 \*/* **void** onError(**int** errorCode);  
}

## removeListener(ClimateChangeListener listener)

移除添加的listener。

## clearListeners()

清空listener集合。

# com.ford.media.MediaEventManager

## USB广播

增加了USB扫描失败、设备不支持的广播Action，其余的USB插入、拔出、扫描等，使用原生广播Action即可。用法与原生保持一致。

*/\*\*  
 \* Broadcast Action: The media scanner has failed to scan a directory.  
 \* The path to the failure scanned directory is contained in the Intent.mData field.  
 \*/***public static final** String ***ACTION\_MEDIA\_SCANNER\_SCAN\_FAILED*** = **"ford.intent.action.MEDIA\_SCANNER\_SCAN\_FAILED"**;  
  
*/\*\*  
 \* Broadcast Action: The device is not supported  
 \* The path to the mount point for the unsupport device is contained in the Intent.mData field.  
 \*/***public static final** String ***ACTION\_MEDIA\_UNSUPPORT*** = **"ford.intent.action.MEDIA\_UNSUPPORT"**;

## public static FordMediaEventManager getInstance(Context context)

获取单例

## void sendVRTips(String text)

发送VR提示文本给车机底部导航栏显示。

## void sendID3Info(ID3Info id3Info)

将随心听播放的ID3信息，发送给仪表显示。ID3Info的字段定义，与百度《随心听交互协议\_3.8.pdf》一致。

## void sendPlayState(int state)

将随心听播放状态，发送给仪表显示。state定义，

**public static final int *PLAY\_STATE\_BUFFERING*** = 1; *//buffering***public static final int *PLAY\_STATE\_PLAYING*** = 2; *//play***public static final int *PLAY\_STATE\_PAUSE*** = 3; *//pause***public static final int *PLAY\_STATE\_STOP*** = 4; *//stop*

## void sendPlayMode(int mode)

将随心听播放模式，发送给仪表显示。mode定义：

**public static final int *PLAY\_MODE\_SINGLE\_CYCLE*** = 0; *//single cycle***public static final int *PLAY\_MODE\_SHUFFLE\_PLAY*** = 1; *//shuffle play***public static final int *PLAY\_MODE\_LIST\_LOOP*** = 2; *//list loop***public static final int *PLAY\_MODE\_ORDER\_PLAY*** = 3; *//order play*

## addListener(ActionListener listener)

增加listener。

## removeListener(ActionListener listener)

移除listener。

## clearListeners()

清空listener。

## 回调接口

**public interface** ActionListener {  
 **void** onAction(**int** action);  
}

当用户通过仪表控制多媒体时，会回调onAction方法，action参数定义：

**public static final int *ACTION\_PLAY*** = 0;*//播放***public static final int *ACTION\_PAUSE*** = 1;*//暂停***public static final int *ACTION\_PREVIOUS*** = 2;*//上一曲***public static final int *ACTION\_NEXT*** = 3;*//下一曲*

# com.ford.camera.EVSCameraManager

图像evs接口，请阅读《evc\_camera-api\_description-v2.1-20200303172110-kkovev.pdf》接口文档。

# com.ford.vendor.FordEnhanceMemoryManager

em接口，按之前邮件excel表格定义的，请参考邮件。

## void changeDriverProfile(int index)

形参：驾驶员档案索引，合法值大于等于0，

驾驶员档案索引（1-4：用户创建的档案

5：访客档案）  
 hmi切换驾驶员档案

## interface IOnProfileActivebyHmi{

void onProfileActive(int index);

形参：驾驶员档案索引，合法值大于等于0，

驾驶员档案索引（1-4：用户创建的档案

5：访客档案）

}hmi切换驾驶员档案的回调

## interface IOnProfileActivebyKey{

void onProfileActive(int index);

形参：驾驶员档案索引，合法值大于等于0

驾驶员档案索引（1-4：用户创建的档案

5：访客档案）

} 车钥匙、手机钥匙切换驾驶员档案的回调

## boolean getGlobalProperty(int propId);

形参1：智能手机钥匙连接状态属性对应的id

返回值1：智能手机钥匙连接状态  
 功能：获取属性id对应的值

## void createDriverProfile(int index);

形参：驾驶员档案索引，合法值大于等于0

驾驶员档案索引（1-4：用户创建的档案

5：访客档案）  
 创建驾驶员档案

## interface IOnProfileCreate{

void onProfileCreate(int index);

形参：驾驶员档案索引，合法值大于等于0

驾驶员档案索引（1-4：用户创建的档案

5：访客档案）

} 创建驾驶员档案的回调

## interface IPaakConnectionChanged{

void onPaakConnectionChanged(byte state);

形参列表：

state：智能手机钥匙连接状态值，2：连接，其他值：未连接

}  
 监听智能手机钥匙连接状态

## void requestPhonePairing(int profileIndex);

形参列表：profileIndex:

驾驶员档案索引（1-4：用户创建的档案5：访客档案）  
驾驶员档案关联智能手机钥匙 前提：智能手机钥匙已连接

## void requestPhoneUnPairing(int profileIndex);

形参列表：profileIndex：

驾驶员档案索引（1-4：用户创建的档案5：访客档案）  
驾驶员档案取消关联智能手机钥匙

## void requestPhoneOverwritePairing(int profileIndex);

形参列表：profileIndex:

驾驶员档案索引（1-4：用户创建的档案5：访客档案）  
驾驶员档案强制关联智能手机钥匙 前提：智能手机钥匙已经被占用

## interface PhoneAssoProfileCallback{

void onChangeEvent(int eventType,int action,int index);

形参列表：eventType：

0：智能手机钥匙关联档案，1：智能手机钥匙取消关联档案，2：中断智能手机钥匙关联档案，3：已占用智能手机钥匙关联档案

action：

当eventType=0（智能手机钥匙关联档案）时，0:手机钥匙被占用的回调，1:手机钥匙关联成功的回调，2:手机钥匙关联失败的回调；3:关联错误设备的回调。

当eventType=1（智能手机钥匙取消关联档案）时，0:手机钥匙取消关联档案的回调。

当eventType=2（中断智能手机钥匙关联档案）时，0:中断智能手机钥匙关联档案的回调。

当eventType=3（已占用智能手机钥匙关联档案）时，0:已占用智能手机钥匙关联档案的回调。

index：驾驶员档案索引（1-4：用户创建的档案5：访客档案）

} 智能手机钥匙关联/取消关联档案的回调

## void requestCarKeyPairing(int profileIndex);

形参列表：profileIndex:档案索引，

驾驶员档案索引（1-4：用户创建的档案5：访客档案）   
驾驶员档案关联车钥匙

## void requestCarKeyUnPairing(int profileIndex);

形参列表：profileIndex:档案索引

驾驶员档案索引（1-4：用户创建的档案5：访客档案）  
档案取消关联车钥匙

## void requestCarKeyOverwritePairing(int profileIndex);

形参列表：profileIndex:档案索引，

驾驶员档案索引（1-4：用户创建的档案5：访客档案）  
驾驶员档案强制关联车钥匙 前提：智能车钥匙已经被占用

## interface CarKeyAssoProfileCallback{

onChangeEvent(int eventType,int action,int index);

形参列表：eventType：

0：车钥匙关联档案，1：车钥匙取消关联档案，2：中断车钥匙关联档案，3：已占用车钥匙关联档案

action：当eventType=0（车钥匙关联档案）时，0:车钥匙被占用的回调，1:车钥匙关联成功的回调，2:车钥匙关联失败的回调；3:关联错误设备的回调。

当eventType=1（车钥匙取消关联档案）时，0:车钥匙取消关联档案的回调。

当eventType=2（中断车钥匙关联档案）时，0:中断车钥匙关联档案的回调。

当eventType=3（已占用车钥匙关联档案）时，0:已占用车钥匙关联档案的回调。

index：

驾驶员档案索引（1-4：用户创建的档案5：访客档案）

} 驾驶员档案关联/取消关联车钥匙的回调

## byte[] getProfilesKeyFobAssociated();

返回值：返回3个个性化档案的与车钥匙的关联状态，byte[0]=0，代表第一个档案与车钥匙无关联，

byte[0]=1，代表第一个档案与车钥匙有关联  
获取所有个性化档案的与车钥匙的关联状态

## byte[] getProfilesKeyPhoneAssociated();

返回值：返回3个个性化档案的与手机钥匙的关联状态，byte[0]=0，代表第一个档案与手机钥匙无关联，byte[0]=1，代表第一个档案与手机钥匙有关联  
获取所有个性化档案的与手机钥匙的关联状态

## interface IOnProfilesDelete{

void onProfilesDelete(int[] indexes);

形参：删除的驾驶员档案索引的数组：

合法值大于等于0，

驾驶员档案索引（1-4：用户创建的档案5：访客档案）

} 批量删除驾驶员档案的回调

## void requestDeleteProfiles(int[] indexes)

形参列表：indexes:

驾驶员档案索引的数组，

驾驶员档案索引（1-4：用户创建的档案 5：访客档案）  
批量删除驾驶员档案

## void stopCarKeyPairing(int profileIndex);

形参列表：profileIndex:

档案索引，驾驶员档案索引（1-4：用户创建的档案 5：访客档案）  
中断驾驶员档案关联车钥匙

## void stopPhonePairing(int profileIndex);

形参列表：profileIndex:

档案索引，驾驶员档案索引（1-4：用户创建的档案 5：访客档案）  
中断驾驶员档案关联智能手机钥匙

## interface IOnGearChanged{

void onGearChanged(int gear);

形参列表：当前档位：

GEAR\_NEUTRAL = 0;

GEAR\_PARKING = 1;

GEAR\_REVERSE = 2;

GEAR\_DRIVE = 3.

} 档位变化的回调

## int getGear();

返回值：当前档位：

GEAR\_NEUTRAL = 0;

GEAR\_PARKING = 1;

GEAR\_REVERSE = 2;

GEAR\_DRIVE = 3.  
获取档位

## List<String> getCollectRadios()

收藏电台的读取，最多18个电台

## void setCollectRadios(List<String> radios)

形参：收藏电台的集合  
收藏电台的设置

## interface ICollectedRadiosChange{

void onCollectedRadioChange(List<String> radios);

形参：收藏电台的集合

} 收藏电台变化的回调

## List<String> getAutoSavedRadios() 自动存储电台的读取，最多18个电台

## void setAutoSavedRadios(List<String> radios)

形参：自动存储电台的集合  
自动存储电台的设置

## interface IAutoSavedRadiosChange{

void onAutoSavedRadiosChange(List<String> radios);

形参：自动存储电台的集合

} 自动存储电台变化的回调

## int getTheme() 主题的读取

## void setTheme(int theme) 主题的设置

## interface IThemeChange{

void onThemeChange(int theme);

} 主题变化的回调

## int getDistanceUnit() 获取距离单位

## void setDistanceUnit(int unit) 设置距离单位

## interface IDistanceUnitChange{

void onDistanceUnitChange(int unit);  
距离单位的回调

## int getTemperatureUnit() 获取温度单位

## void setTemperatureUnit(int unit) 设置温度单位

## interface ITemperatureUnitChange{

void onTemperatureUnitChange(int unit);  
温度单位的回调

## int getTyreUnit() 获取胎压单位，默认值kPa

## void setTyreUnit(int tyreUnit) 设置胎压单位

## interface ITyreUnitChange{

void onTyreUnitChange(int tyreUnit);  
胎压单位的回调

## int getMeasureUnit() 获取测量单位，默认值是km&km/l

## void setMeasureUnit(int measureUnit ) 设置测量单位

## interface IMeasureUnitChange{

void onMeasureUnitChange(int measureUnit);  
测量单位的回调

## int getAmbientLampSwitch() 获取氛围灯开关

## void setAmbientLampSwitch(int switch) 设置氛围灯开关

## interface IAmbientLampSwitchChange{

void onAmbientLampSwitchChange(int swicth);  
氛围灯开关变化的回调

## int getAmbientColor() 获取氛围灯颜色

## void setAmbientColor(int color ) 设置氛围灯颜色

## interface IAmbientColorChange{

void onAmbientColorChange(int color);  
氛围灯颜色变化的回调

## 10.51 int getAmbientBrightness() 获取氛围灯亮度

## void setAmbientBrightness(int brightness) 设置氛围灯亮度

## interface IAmbientBrightnessChange{

void onAmbientBrightnessChange(int brightness);  
氛围灯亮度变化的回调

## boolean getReverseImageDelay() 获取倒车影像延迟

## void setReverseImageDelay(boolean delay) 设置倒车影像延迟

## interface IReverseImageDelayChange{

void onReverseImageDelayChange(boolean delay);

}倒车影像延迟变化的回调

## int getSpeedVolCompensation() 获取车速音量补偿设置

## void setSpeedVolCompensation(int compensation) 设置车速音量补偿设置

## interface ISpeedVolCompensationChange{

void onSpeedVolCompensationChange(int compensation);  
车速音量补偿设置变化的回调

## int getAirConTemperature() 获取空调温度

## void setAirConTemperature(int temperature) 设置空调温度

## interface IAirConTemperatureChange{

void onAirConTemperatureChange(int temperature);  
空调温度变化的回调

## void setDriveMode(int mode) 设置驾驶模式

# com.ford.vendor.FordEVChargeManager

EV接口，按之前邮件excel表格定义的，请参考邮件。

## byte getPlugConnectStatus();

获取车与充电桩的连接状态

## interface IPlugConnectStatus{

void onPlugConnectStatus(boolean status){

status:true：连接

false：未连接}

车与充电桩的连接状态的回调

## boolean isChargePlanPos();

判断车辆位置是否在充电计划的位置

返回值：当前位置是否在充电计划位置的状态，

true：当前位置在充电计划内；

false：当前位置不在充电计划内

## IVehiclePosChange {

void onVehiclePosChange(byte posId){

posId：Saved Charge Location ID，0：不在收藏充电位置，1-10：在收藏的充电位置

}   
车辆位置变化的回调

## boolean getChargPlanSwitch();

返回值：获取充电计划的开关，

true：开启，false：关闭

获取车辆位置充电计划的开关

前提：1.当前位置在充电计划的位置，否则返回false

## IChargPlanSwitchChange{

void onChargPlanSwitchChange(boolean on);

on：充电计划开关

}

车辆位置充电计划的开关变化的回调

前提：用户触发充电计划的开关

## void setChargPlanSwitch(boolean on);

形参:充电计划的开关，true：开启，false：关闭

打开或者关闭当前位置的充电计划

## byte getChargePercentage();

返回值：电池当前电量百分比值，单位为“%”，值为截尾数据，例：97.5% -> “97%”

获取电池当前电量百分比值

## interface IChargeProcessChange{

void onChargeProcessChange(int process){

process:电池当前电量百分比值，单位为“%”，值为截尾数据，例：97.5% -> “97%”。

}

电池当前电量百分比值变化的回调

## ChargePlan getCurentChargePlan();

返回值：获取当前充电计划，充电计划见sheet-ChargePlan

获取当前充电计划

前提：1.当前位置在充电计划的位置，否则，返回null

## byte getChrgStat();

返回值：0: Not Ready,7:charging

获取车的充电状态

## interface IChrgStatChange{

void onChrgStatChange(byte state){

state:车的充电状态

}

车的充电状态的回调

## int getRemainingMileage();

返回值：剩余里程值，单位为“km”

获取剩余里程，若有CE-DTE的数据，优先展示CE-DTE为剩余里程，

否则，展示普通DTE

## float getfullyChargeHours();

返回值：充满电预测所用小时数，单位是小时，取数字小数点后一位

获取充满电预测所用小时数

前提：无连接充电桩\在已收藏充电位置\无使用或打开充电计划\电池电量<充电目标电量，否则返回0.0

## byte getChargeTargetPercent();

返回值：充电的目标电量百分比，单位为“%”

获取充电的目标电量百分比

前提：当前位置在充电计划的位置\充电计划开启，返回充电计划的设置数据，否则返回默认值；

## ChargeStartTime getChargeStartTime();

形参：充电开始时间，

获取充电开始时间

## ChargeEndTime getChargeEndTime();

返回值：充电结束时间，

获取充电结束时间

## DepartAndComfort getDepartAndComfort();

返回值：出发时间及温度水平，DepartAndComfort

获取最近的出发时间及温度水平

前提：出发时间设置开关开启且设置了出发时间，返回最近的出发时间，否则返回null；

## boolean isVehicleRun();

返回值：获取车辆的运转状态，true：运行；false：不运行

获取车辆的运转状态（点火开关）

## interface IVehicleRunChange{

void onVehicleRunChange(boolean state){

}

车辆的运转状态的回调

## List<ChargeLocation> getUnsavedChargeLocations();

返回值：未保存的历史充电位置的列表，至多返回最近的10个充电位置信息，按时间倒序排列

获取未保存地址列表

## List<ChargePlan> getChargePlans();

返回值：充电计划列表，至多最近的10个充电计划信息，

按时间倒序排列

获取充电计划列表

## void insertChargePlan(ChargePlan chargePlan);

形参：充电计划  
 插入一个充电计划

## void updateChargePlan(ChargePlan chargePlan);

形参：充电计划  
 更新一个充电计划

## void deleteChargePlan(int id);

形参：id:：充电计划id，等于已收藏充电位置id，

删除一个充电计划

## Interface IEditChargePlan{

void onEditChargePlan(byte eventType,boolean state,int id);

形参列表：

eventType：

0：插入，1:更新，2：删除

state:编辑一个充电计划的操作是否成功

id：充电计划在车底层数据库的唯一标示

}

更新\插入\删除一个充电计划的回调

## List<DepartAndComfort> getDepartAndComforts();

返回值：一周的出发时间和温度水平的数据，大小0-14

获取所有出发时间和温度水平的数据

## void setDepartAndComfortSwitch(boolean on);

形参:出发与温度水平的开关，true：开启，false：关闭

打开或者关闭出发时间和温度水平功能

## IDepartAndComfortSwitchChange{

void onDepartAndComfortSwitchChange(boolean on);

on：出发时间和温度水平的功能开关

}

打开或者关闭出发时间和温度水平功能的回调

## void saveDepartAndComforts(List<DepartAndComfort> departAndComforts);

形参：出发时间和温度水平的数据，

保存出发时间和温度水平的数据（一个或者多个）

## void deleteDepartAndComforts(List<byte> ids);

形参：DepartAndComfort.id构成的list，

删除出发时间和温度水平的数据（一个或者多个）

## void deleteAllDepartAndComforts();

形参：删除一周中所有出发时间和温度水平的数据

删除所有出发时间和温度水平的数据

## IEditDepartAndComfort {

void onEditDepartAndComfort(byte evenType,boolean state);

evenType：

0：保存，1：删除。

state：保存、删除出发时间和温度水平的操作是否成功

保存、删除出发时间和温度水平的数据的回调

## interface IChargeAndDepartConflict{

void onChargeAndDepartConflict(int type);

0:Charging outside time window（在充电计划预设时间段外充电）

1:Not reaching desired level（无法完成充电至目标电量）

}

冲突提醒的回调

## void disConnection() 取消相关service的连接

## interface IRemainMileageListener{

void onRemainingMileage(int mileage){

mileage:剩余里程值

}

剩余里程值回调

## byte getBattLowWarning();

返回值：0: NULL,1:LoDte,2:ZeroDteDepletedBattery,3:Not used  
获取车辆低电池警告

## interface IBattLowWarning{

void onBattLowWarning(int type);

0:NULL

1:LoDte（低电量）

2:ZeroDteDepletedBattery (电量为0)

3:Not used

} 车辆低电池警告回调

## byte getChrgrInPwType()

获取充电功率

## interface IChrgrInPwTypeChange

**{**

**void onChrgrInPwTypeChange(byte type);**

**}**

充电功率的回调

## interface IRemainingMileageChange

**{**

**void onRemainingMileageChange(int mileage);**

**}**

电池续航里程的回调

## interface IChargeEndTimeChange

**{**

**void onChargeEndTimeChange(byte hours);**

**}**

预计充电时间（充电结束时间）的回调

# com.ford.vendor.FordBluetoothManager

提供监听蓝牙状态的接口，蓝牙的地址、通话状态等，均从回调接口的SVCallInfo中取出即可。

**void** addListener(IBtHfpListener listener)

*/\*\*  
 \* 通话状态变化回调  
 \*/***public interface** IBtHfpListener {  
 **void** onPhoneCallStateChange(SVCallInfo svCallInfo);  
}

通话状态：

**static class** CallState {  
 **public static final int *CALL\_STATE\_ACTIVE*** = 0;  
 **public static final int *CALL\_STATE\_HELD*** = 1;  
 **public static final int *CALL\_STATE\_DIALING*** = 2;  
 **public static final int *CALL\_STATE\_ALERTING*** = 3;  
 **public static final int *CALL\_STATE\_INCOMING*** = 4;  
 **public static final int *CALL\_STATE\_WAITING*** = 5;  
 **public static final int *CALL\_STATE\_HELD\_BY\_RESPONSE\_AND\_HOLD*** = 6;  
 **public static final int *CALL\_STATE\_TERMINATED*** = 7;13  
 **public static final int *CALL\_STATE\_MISSCALL*** = 8;  
}

# com.ford.vendor.FordCeDTEManager

## void setTripInfoStructure(byte[] tripInfoStructure)

传数据给Msg

## void setTripStatus(byte[] tripStatus)

传数据给Msg

## void setConnectorType(byte[] connectorType)

传数据给MSG

## void setBatteryPackType(byte[] batteryPackType)

传数据给MSG

## void onConnect(FordBaseManager.ConnectionCallback connection) 连接服务，连接后同步返回数据后才有信号数据返回

## void onDisConnect() 断开服务连接

## int getRemainingMileage()

获取剩余里程

# com.ford.ea.FordEmergencyManager

## void sendReqWithRsp(BaseReqMessage baseReqMessage)

发送紧急救援相关的参数并带返回消息

void sendRequestNoRsp(BaseReqMessage baseReqMessage)

发送紧急救援相关的参数不带返回消息

BaseReqMessage 救援相关的参数

## int getCrashCan()

获取获取车辆的Crash信号

## int getIgnKeyType\_D\_Actl()

获取车辆的mykey的使用信号

## *int getMyKey\_e911Override\_St()*

获取车辆的mykey的override状态信号

## int getDrStatDrv\_B\_Actl()

获取车辆开门信号DrStatDrv\_B\_Actl

## int getIgnition\_Status()

获取车辆点火信号Ignition\_Status

## int getPOWER\_MODE()

获取车辆电源状态POWER\_MODE的信号

## int getVEDS()

获取车辆发生crash时的车辆的信息信号

## public interface IRspCallback {

void changed(EmergencyAssistanceResponse emergencyAssistanceResponse);  
}

# com.ford.vendor.FordBtManager

## void connect(String address)

连接蓝牙

address 蓝牙地址

## void dial(String address, String number)

拨打电话

address 蓝牙地址 number 电话号码

## void terminateCall(String address)

挂断电话

Address蓝牙地址

## void connectAudio(String address)

连接语音

Address 蓝牙地址

## void disconnectAudio(String address)

断开语音

Address 蓝牙地址

## interface IBtHfpChangeListener {

void onHfpDeviceConnectedStatusChange(String address, int state);  
 蓝牙连接状态的回调

Address 蓝牙地址

State 连接状态  
 void onPhoneCallAudioStatusChanged(String address, int status);

语音连接状态的回调

Status 连接状态  
 void onPhoneCallStateChange(SVCallInfo callInfo);

电话连接变化的回调

SVCallInfo 电话相关信息  
}

# com.ford.audio. FordAudioManager

## public static final String CAR\_AUDIO\_TYPE\_EA=

SvCarAudioManager.CAR\_AUDIO\_TYPE\_EA;

public static final String CAR\_AUDIO\_TYPE\_EA\_PHONE = SvCarAudioManager.CAR\_AUDIO\_TYPE\_EA\_PHONE;

public static final String CAR\_AUDIO\_TYPE\_EA\_PROMPT =SvCarAudioManager.CAR\_AUDIO\_TYPE\_EA\_PROMPT;

对EA audio接口有以下几点说明：

1、 申请/释放焦点采用原生的接口，不过还需要适配下car\_audio\_type

2、 在播放合成音时，new AudioTrack 必须将申请焦点时定义的AudioAttributes传进来。

3、 开始播放合成音，调用AudioTrack play，当合成音播放完时要调用AudioTrack的stop方法。

# com.ford.audio.FordCommonManager

## void connect(ConnectionCallback connection) 启动服务，并同步将消息发送出去

## void disConnect()

断开服务

## sendReqWithRsp(BaseReqMessage baseReqMessage)

发送带返回消息的接口 BaseReqMessage 救援相关的参数

## sendRequestNoRsp(BaseReqMessage baseReqMessage) 发送不带返回消息的接口 BaseReqMessage 救援相关的参数

## void setResponseListener(IResponseCallback responseListener)

设置数据回调

## interface IResponseCallback {

void changed(int commandid, BaseResponse baseResponse);

} can信号得返回接口

# com.ford.vendor.FordHardKeyManager

## void registerKeyEvent(int AppID, IkeyEventListener listener); 根据相应的AppID注册物理按键

AppID: //自定义按键模块AppID

//本地收音机

public static final int APP\_ID\_RADIO = DesayHardKeyManager.APP\_ID\_RADIO;

//蓝牙音乐

public static final int APP\_ID\_BT\_MUSIC = DesayHardKeyManager.APP\_ID\_BT\_MUSIC;

//随心听

public static final int APP\_ID\_BAIDU\_RADIO = DesayHardKeyManager.APP\_ID\_BAIDU\_RADIO;

//蓝牙电话

public static final int APP\_ID\_BT\_PHONE = DesayHardKeyManager.APP\_ID\_BT\_PHONE;

//语音

public static final int APP\_ID\_VR = DesayHardKeyManager.APP\_ID\_VR;

//设置

public static final int APP\_ID\_SETTINGS = DesayHardKeyManager.APP\_ID\_SETTINGS;

//电源管理

public static final int APP\_ID\_POWER\_MANAGEMENT = DesayHardKeyManager.APP\_ID\_POWER\_MANAGEMENT;

//导航

public static final int APP\_ID\_NAVIGATION = DesayHardKeyManager.APP\_ID\_NAVIGATION;

//rvc

public static final int APP\_ID\_RVC = DesayHardKeyManager.APP\_ID\_RVC;

//EA

public static final int APP\_ID\_EA = DesayHardKeyManager.APP\_ID\_EA;

listener见19.3

## void unregisterKeyEvent(int AppID, IkeyEventListener listener)

**取消物理按键的注册 Appid见19.1 listener见19.3**

## interface IkeyEventListener

**{**

**void onCustomKeyEvent(int keyCode, int keyAction, String value);**

**}**

**监听物理按键的回调  
 相关参数如下：**

//自定义KeyCode

//按键定义如下，对应MRD（Ford\_MRD\_Switch\_Input\_Matrix\_V08.xlsx）中的label列：

/\* SWCs (CAN)方向盘按键 \*/

public static final int KEYCODE\_SWC\_SEEK\_MINUS = DesayHardKeyManager.KEYCODE\_SWC\_SEEK\_MINUS;//Seek-

public static final int KEYCODE\_SWC\_SEEK\_PLUS = DesayHardKeyManager.KEYCODE\_SWC\_SEEK\_PLUS;//Seek +

public static final int KEYCODE\_SWC\_SEEK\_LEFT\_PHONE = DesayHardKeyManager.KEYCODE\_SWC\_SEEK\_LEFT\_PHONE;//Seek Left + Phone "SEND"

public static final int KEYCODE\_SWC\_SEEK\_RIGHT\_PHONE = DesayHardKeyManager.KEYCODE\_SWC\_SEEK\_RIGHT\_PHONE;//Seek Right + Phone "End"

public static final int KEYCODE\_SWC\_PUSH\_TO\_TALK = DesayHardKeyManager.KEYCODE\_SWC\_PUSH\_TO\_TALK;//Push to Talk (Voice)(PTT)

public static final int KEYCODE\_SWC\_PHONE\_SEND\_END = DesayHardKeyManager.KEYCODE\_SWC\_PHONE\_SEND\_END;//Phone "Send&End"

public static final int KEYCODE\_SWC\_MENU = DesayHardKeyManager.KEYCODE\_SWC\_MENU;//MENU

public static final int KEYCODE\_SWC\_DISP = DesayHardKeyManager.KEYCODE\_SWC\_DISP;//DISP

public static final int KEYCODE\_SWC\_AUDIO = DesayHardKeyManager.KEYCODE\_SWC\_AUDIO;//Audio

public static final int KEYCODE\_SWC\_SETTING = DesayHardKeyManager.KEYCODE\_SWC\_SETTING;//Setting

public static final int KEYCODE\_SWC\_NAVIGATION = DesayHardKeyManager.KEYCODE\_SWC\_NAVIGATION;//Navigation

/\* Mini ICP (LIN) 中控按键\*/

public static final int KEYCODE\_ICP\_POWER\_VOLUME = DesayHardKeyManager.KEYCODE\_ICP\_POWER\_VOLUME;//Power Volume

public static final int KEYCODE\_ICP\_DAT = DesayHardKeyManager.KEYCODE\_ICP\_DAT;//DAT Shortcut

public static final int KEYCODE\_ICP\_AUDIO\_POWER\_ONOFF = DesayHardKeyManager.KEYCODE\_ICP\_AUDIO\_POWER\_ONOFF;//Audio "Powor On/Off"

public static final int KEYCODE\_ICP\_SEEK\_LEFT = DesayHardKeyManager.KEYCODE\_ICP\_SEEK\_LEFT;//Seek Left

public static final int KEYCODE\_ICP\_SEEK\_RIGHT = DesayHardKeyManager.KEYCODE\_ICP\_SEEK\_RIGHT;//Seek Right

public static final int KEYCODE\_ICP\_PLAY\_PAUSE = DesayHardKeyManager.KEYCODE\_ICP\_PLAY\_PAUSE;//Play / Pause icon

public static final int KEYCODE\_ICP\_REAR\_AUDIO\_LOCK = DesayHardKeyManager.KEYCODE\_ICP\_REAR\_AUDIO\_LOCK;//Rear Audio Lock

public static final int KEYCODE\_ICP\_PARKING\_CAMERA = DesayHardKeyManager.KEYCODE\_ICP\_PARKING\_CAMERA;//Parking+Camera Hot Key

public static final int KEYCODE\_ICP\_SOUND\_MENU = DesayHardKeyManager.KEYCODE\_ICP\_SOUND\_MENU;//Sound Menu

public static final int KEYCODE\_ICP\_SOURCE\_ALL\_MEDIA = DesayHardKeyManager.KEYCODE\_ICP\_SOURCE\_ALL\_MEDIA;//Source (All MEDIA)

public static final int KEYCODE\_ICP\_DISPLAY = DesayHardKeyManager.KEYCODE\_ICP\_DISPLAY;//Display

/\* RCCM CCH/EFP \*/

// 暂时无定义

/\* RACM (CAN)后排按键 \*/

public static final int KEYCODE\_RACM\_SOURCE = DesayHardKeyManager.KEYCODE\_RACM\_SOURCE;//Source

public static final int KEYCODE\_RACM\_SEEK\_FORWARD = DesayHardKeyManager.KEYCODE\_RACM\_SEEK\_FORWARD;//Seek Forward

public static final int KEYCODE\_RACM\_SEEK\_BACK = DesayHardKeyManager.KEYCODE\_RACM\_SEEK\_BACK;//Seek Back

//自定义Key的KeyAction

public static final int KEYACTION\_UP = DesayHardKeyManager.KEYACTION\_UP; //短键的up事件

public static final int KEYACTION\_DOWN = DesayHardKeyManager.KEYACTION\_DOWN; //短按的down事件

public static final int KEYACTION\_LONG\_UP = DesayHardKeyManager.KEYACTION\_LONG\_UP; //长按的up事件

public static final int KEYACTION\_LONG\_DOWN = DesayHardKeyManager.KEYACTION\_LONG\_DOWN; //长按的down事件

public static final int KEYACTION\_ERROR = DesayHardKeyManager.KEYACTION\_ERROR; //错误事件

# com.ford.car.FordCarClimmateManager

## int getGlobalProperty(int type)

当前时刻的PM2.5值

参数.int type:为要获取数据类型

TYPE\_CCM\_MESSAGE\_MISSING=0--空调发出的携带PM2.5传感器（滤芯）数据的CAN信号是否有效

TYPE\_PM\_MESSAGE\_MISSING=1--空调开启和关闭触发的CAN信号是否有效

TYPE\_PMCABN\_CONC\_ACTL=2--当前PM2.5的值

TYPE\_PMSNSCABN\_D\_STAT=3--PM2.5传感器的状态

## interface AARChangeListener

**{**

**void onChangeEvent(int eventType, int[] intValues, float[] floatValues);**

**}  
当前时刻PM2.5变化的回调**

参数说明：

1.int eventType

TYPE\_CCM\_MESSAGE\_MISSING=0x00--空调发出的携带PM2.5传感器（滤芯）数据的CAN信号是否有效

TYPE\_PM\_MESSAGE\_MISSING=0x01--空调开启和关闭触发的CAN信号是否有效

TYPE\_PMCABN\_CONC\_ACTL=0x02--当前PM2.5的值

TYPE\_PMSNSCABN\_D\_STAT=0x03--PM2.5传感器的状态

2.int[] intValue:

intValue[0]:几个类型信号的值

## int getGlobalProperty1(int type)

X分钟前的PM2.5值

参数：int type:为要获取数据类型

TYPE\_PMCABN02MNTE\_CONC\_ACTL=0x10--2分钟之前的PM2.5值

TYPE\_PMCABN04MNTE\_CONC\_ACTL=0x11--4分钟之前的PM2.5值

TYPE\_PMCABN06MNTE\_CONC\_ACTL=0x12--6分钟之前的PM2.5值

TYPE\_PMCABN08MNTE\_CONC\_ACTL=0x13--8分钟之前的PM2.5值

TYPE\_PMCABN10MNTE\_CONC\_ACTL=0x14--10分钟之前的PM2.5值

TYPE\_PMCABN12MNTE\_CONC\_ACTL=0x15--12分钟之前的PM2.5值

TYPE\_PMCABN14MNTE\_CONC\_ACTL=0x16--14分钟之前的PM2.5值

TYPE\_PMCABN16MNTE\_CONC\_ACTL=0x17--16分钟之前的PM2.5值

TYPE\_PMCABN18MNTE\_CONC\_ACTL=0x18--18分钟之前的PM2.5值

TYPE\_PMCABN20MNTE\_CONC\_ACTL=0x19--20分钟之前的PM2.5值

## void setGlobalProperty(int type, int value)

发送当前空气过滤状态

参数：

1.int type:TYPE\_PMCABN\_D\_STAT=0x20--发送空气状态

2.int vale：空气状态

发送当前PM2.5的等级

1.int type:TYPE\_PMCABNLVL\_D\_STAT=0x21--发送PM2.5的等级

2.int vale:PM2.5的等级

## void setNewAirRefresh()

开启/关闭座舱新风按钮

## int getNewAirRefresh()

获取座舱新风按钮可操作状态

返回值：

NEW\_AIR\_ENABLED\_INACTIVE = 0--座舱新风未激活

NEW\_AIR\_ACTIVE = 1--座舱新风已启动

NEW\_AIR\_DISABLED = 2--座舱新风已关闭

NEW\_AIR\_UNUSED = 3 --座舱新风不可用

## int getAcConfig()

空调是否可用

返回值：

HAVAC\_CONFIG\_NOT\_AVAILABLE = 0--空调不可用

HAVAC\_CONFIG\_AVAILABLE = 1--空调可用

# com.ford.vendor.FordElectronicHorizonManager

## void onConnect(FordBaseManager.ConnectionCallback connection)

连接服务，同步数据返回接口，连接后才有数据返回

## void onDisConnect()

断开服务连接

## ElectronicHorizon1\_Rsp getElectronicHorizon1\_Rsp()

发送 Adas EH1 数据内容

## ElectronicHorizon2\_Rsp getElectronicHorizon2()

发送 Adas EH2 数据内容