ZH SDK Second-generation Berry protocol update change instructions

Table of Contents

1. Distinguish device protocol version

1.1. Bluetooth Scanning

- Call to start scanning devices and set callback

```
/**
* Start scanning devices
*/
ControlBleTools.getInstance().startScanDevice(ScanDeviceCallBack)
```

- Processing callback function

```
ScanDeviceCallBack.onBleScan(ScanDeviceBean)
```

- Determine the protocol version through **ScanDeviceBean.protocolName**

```
BleCommonAttributes.DEVICE_PROTOCOL_APRICOT //The old protocol is referred to as
APRICOT
BleCommonAttributes.DEVICE_PROTOCOL_BERRY //The second generation new protocol is
referred to as BERRY
```

1.2. Use system LE Bluetooth scanning

- Call to start system LE scanning device and set callback

```
var mLeScanCallback: LeScanCallback = LeScanCallback { device:BluetoothDevice,
rssi:Int, scanRecord:ByteArray ->
      val scanResult = ScanResult(device, ScanRecord.parseFromBytes(scanRecord), rssi,
SystemClock.elapsedRealtimeNanos())
      val scanDeviceBean = ScanDeviceBean(scanResult)
}
```

- Determine the protocol version through **ScanDeviceBean.protocolName**

1.3. Scan the QR code

- APP scans the bound QR code on the device and obtains the QR code content string **qrCode**
- Construct a ScanQrDeviceBean object through qrCode

```
val scanQrDeviceBean:ScanQrDeviceBean = ScanQrDeviceBean(qrCode)
```

- Determine the protocol version through **ScanQrDeviceBean.scanDeviceBean.protocolName**

```
BleCommonAttributes.DEVICE_PROTOCOL_APRICOT //The old protocol is referred to as
APRICOT
BleCommonAttributes.DEVICE_PROTOCOL_BERRY //The second generation new protocol is
referred to as BERRY
```

TIP **BERRY scans the QR code to add protocolName to distinguish the protocol version. For devices that did not have this field before, all are APRICOT**

2. Connect

2.1. APRICOT connection process

- The APP has completed initializing the SDK, obtaining relevant permissions, and the system Bluetooth is turned on normally
- Call the connection method

```
/**
* Connecting devices
*
* @param name Bluetooth name
* @param address Bluetooth address
* @deprecated use {@link #connect(String, String, String)} name,
* mac, protocol(ScanDeviceBean.protocolName)
*/
ControlBleTools.getInstance().connect(name,address)
```

- Set the connection status callback

```
/**
* Set the connection status callback
*/
ControlBleTools.getInstance().setBleStateCallBack(BleStateCallBack)
```

- Process the callback result and determine the connection status through **state**

```
BleStateCallBack.onConnectState(state)
// Disconnected
BleCommonAttributes.STATE_DISCONNECTED
// Connecting
BleCommonAttributes.STATE_CONNECTING
// Connected
BleCommonAttributes.STATE_CONNECTED
```

```
// Disconnecting
BleCommonAttributes.STATE_DISCONNECTING
// Connection timed out
BleCommonAttributes.STATE_TIME_OUT
```

2.2. BERRY connection process

- The APP has completed initializing the SDK, obtaining relevant permissions, and the system Bluetooth is turned on normally
- Call the connection method

```
/**
* Connecting devices
*
* @param name Bluetooth name
* @param address Bluetooth address
* @param deviceProtocol device protocol version
* BleCommonAttributes.DEVICE_PROTOCOL_APRICOT,
* BleCommonAttributes.DEVICE_PROTOCOL_BERRY,
* For old devices without this field value, an empty string "" or null can be passed
in, and APRICOT connection is executed by default
*
*/
ControlBleTools.getInstance().connect(deviceName, deviceAddress, deviceProtocol)
```

- Set the connection status callback

```
/**
* Set the connection status callback
*/
ControlBleTools.getInstance().setBleStateCallBack(BleStateCallBack)
```

- Process the callback result and determine the connection status through **state**

```
BleStateCallBack.onConnectState(state)
// Disconnected
BleCommonAttributes.STATE_DISCONNECTED
// Connecting
BleCommonAttributes.STATE_CONNECTING
// Connected
BleCommonAttributes.STATE_CONNECTED
// Disconnecting
BleCommonAttributes.STATE_DISCONNECTING
// Connection timeout
BleCommonAttributes.STATE_TIME_OUT
```

TIP **BERRY connection method adds deviceProtocol device protocol parameter, pass in ScanDeviceBean.protocolName device protocol version obtained by distinguishing device protocol version step (this method is compatible with APRICOT connection)**

3. Binding

3.1. APRICOT Binding Process

- After connecting the device, call the device binding status query interface

```
/**
* Request the binding status of the currently connected device
*/
ControlBleTools.getInstance().requestDeviceBindState(ParsingStateManager.SendCmdStateL
istener)
```

- Set query binding status callback

```
CallBackUtils.setRequestDeviceBindStateCallBack(RequestDeviceBindStateCallBack)
```

- Process the query binding status callback result

```
RequestDeviceBindStateCallBack.onBindState(state)
// Judge by state, false means unbound, you can initiate binding
```

- Call the binding interface

```
/**
* Scan and bind the device (the device needs to confirm the binding)
*/
ControlBleTools.getInstance().bindDevice(ParsingStateManager.SendCmdStateListener)

/**
* Scan the QR code to bind the device (scan the device QR code with verification code,
no device confirmation required)
*
* @param verificationCode is obtained from the QR code information
*/
ControlBleTools.getInstance().bindDevice(verificationCode,ParsingStateManager.SendCmdS
tateListener)
```

- Set the binding callback

```
CallBackUtils.setBindDeviceStateCallBack(BindDeviceStateCallBack)
```

- Process the binding callback result

```
BindDeviceStateCallBack.onDeviceInfo(BindDeviceBean)
//By judging whether BindDeviceBean.deviceVerify is true, the binding is successful,
or judging whether deviceVerifyCode is VerifyCode.SUCCESS, the binding is successful
```

- The App binds the device to the server account. If successful, it sends the user ID to the device and calls the user ID setting interface

```
/**
* Set the binding success user id
*
* @param userId server user id
*/
ControlBleTools.getInstance().sendAppBindResult(userId,
ParsingStateManager.SendCmdStateListener)
```

3.2. BERRY Binding Process

- Complete the device connection and set binding related callbacks

```
CallBackUtils.setBerryBindCallBack(BerryBindCallBack)
```

- Call the interface to set the user id

```
/**
* Set user ID
*
* @param userid userid
* @param phoneName phone model cannot be empty
* @param systemVersion The phone system version cannot be empty
*/
ControlBleTools.getInstance().setUserIdByBerryProtocol(userid, phoneName,
systemVersion,ParsingStateManager.SendCmdStateListener)
```

- Process setting user id callback

```
BerryBindCallBack.onUserIdResult(BerryDeviceInfoBean)
//By judging that BerryDeviceInfoBean.isBind == false is not bound, you can initiate
binding
```

- Call the binding method

```
/**
* Scan and bind the device (the device needs to confirm the binding)
*/
ControlBleTools.getInstance().bindDevice(ParsingStateManager.SendCmdStateListener)
```

- Process the binding callback result

```
BerryBindCallBack.onBindStatu(bindStatus)
//bindStatus --> 0: User clicks to agree 1: Different user ID 2: User clicks to reject
3: Device is already bound
//By judging bindStatus == 0, the binding is successful
```

- The App binds the device to the server account. If successful, the device is notified of successful binding and the binding success response interface is called.

```
/**
 * Notify the device whether the binding is successful
 *
 * @param isSuc Is the binding successful?
 * @param listener
 */
public void
bindDeviceSucByBerryProtocol(isSuc,ParsingStateManager.SendCmdStateListener)
```

- Process the binding success callback result

```
BerryBindCallBack.onBindSuccess(status)
//status --> 0: success; 1: timeout failure
//Status == 0 indicates successful binding
```

TIP

BERRY binding process and related interface modifications, the steps to set the user ID are advanced, the initiation of binding interface no longer distinguishes between scanning or code scanning

4. Connect back

4.1. APRICOT back-connection process

- After connecting the device, call the device binding status query interface

```
/**
```

```
* Request the binding status of the currently connected device
*/
ControlBleTools.getInstance().requestDeviceBindState(ParsingStateManager.SendCmdStateL
istener)
```

- Set query binding status callback

```
CallBackUtils.setRequestDeviceBindStateCallBack(RequestDeviceBindStateCallBack)
```

- Process the query binding status callback result

```
RequestDeviceBindStateCallBack.onBindState(state)
// Judge by state, true means it has been bound and you can proceed to the next step;
false means the device has been unbound and a prompt is displayed saying it has been
unbound
```

- In the bound state, continue to call to verify whether it is the same user interface

```
/**
* Verify the binding user id of the device
*
* @param userid userid
*/
ControlBleTools.getInstance().verifyUserId(userId,
ParsingStateManager.SendCmdStateListener)
```

- Set the verification user ID callback

```
CallBackUtils.setVerifyUserIdCallBack(VerifyUserIdCallBack)
```

- Process the verification user ID callback result

```
VerifyUserIdCallBack.onVerifyState(state)
// If state == 0, it means the same user, and the prompt is that the connection is
successful; if state == 1, it means a different user, and the prompt is bound to
another account
```

4.2. BERRY backlink process

- Complete the device connection and set binding related callbacks

```
CallBackUtils.setBerryBindCallBack(BerryBindCallBack)
```

- Call the interface to set the user id

```
/**
* Set user ID
*
* @param userid User ID
* @param phoneName phone model cannot be empty
* @param systemVersion The phone system version cannot be empty
*/
ControlBleTools.getInstance().setUserIdByBerryProtocol(userid, phoneName,
systemVersion,ParsingStateManager.SendCmdStateListener)
```

- Process setting user id callback

```
BerryBindCallBack.onUserIdResult(BerryDeviceInfoBean)
//By judging BerryDeviceInfoBean.isBind == true, it is bound; false means the device
is unbound, and a prompt is given if it is unbound
// If BerryDeviceInfoBean.isSameUser == true, it means the user is the same, and a
prompt will appear saying that the connection is successful; otherwise, a prompt will
appear saying that the user is bound to another account.
```

TIP **BERRY reconnection process and related interface modifications, compared with APRICOT, the reconnection steps are simplified**

5. Synchronize data

5.1. APRICOT Synchronous Data Flow

- Successfully connect back to the device and call the daily data interface

```
/**
* Get daily data
*/
ControlBleTools.getInstance().getDailyHistoryData(ParsingStateManager.SendCmdStateList
ener)
```

- Set daily data callback

```
CallBackUtils.setFitnessDataCallBack(FitnessDataCallBack)
```

- Handle daily data synchronization progress

```
FitnessDataCallBack.onProgress(progress, total)
```

```
//Progress represents the number of daily data currently synchronized, and total
represents the total number of daily data that need to be synchronized
```

- Process daily data callback results

```
FitnessDataCallBack.onXXXData(XXXBean)
//App stores or uploads XXXBean daily data to the server
```

- Call the interface to obtain motion data

```
/**
* Get device motion data
*/
ControlBleTools.getInstance().getFitnessSportIdsData(ParsingStateManager.SendCmdStateL
istener)
```

- Set the motion data progress callback

```
CallBackUtils.setSportParsingProgressCallBack(SportParsingProgressCallBack)
```

- Processing motion data progress

```
SportParsingProgressCallBack.onProgress(progress, total)
//Progress represents the number of motion data currently synchronized, and total
represents the total number of motion data that need to be synchronized
```

- Set motion data callback

```
CallBackUtils.setSportCallBack(SportCallBack)
```

- Process motion data callback results

```
SportCallBack.onDevSportInfo(DevSportInfoBean)
//App stores or uploads the DevSportInfoBean sports data to the server
```

5.2. BERRY Synchronous Data Process

- Successfully connect back to the device and call the daily data interface

```
/**
* Get daily data
*/
```

```
ControlBleTools.getInstance().getDailyHistoryData(ParsingStateManager.SendCmdStateList
ener)
```

- Set daily data callback

```
CallBackUtils.setFitnessDataCallBack(FitnessDataCallBack)
```

- Handle daily data synchronization progress

```
FitnessDataCallBack.onProgress(progress, total)
//Where progress represents the number of bytes received for daily data, total
represents the total number of bytes of daily data that need to be synchronized
```

- Process daily data callback results

```
FitnessDataCallBack.onXXXData(XXXBean)
//App stores or uploads XXXBean daily data to the server
```

- Call the interface to obtain motion data

```
/**
* Get device motion data
*/
ControlBleTools.getInstance().getFitnessSportIdsData(ParsingStateManager.SendCmdStateL
istener)
```

- Set the motion data progress callback

```
CallBackUtils.setSportParsingProgressCallBack(SportParsingProgressCallBack)
```

- Processing motion data progress

```
SportParsingProgressCallBack.onProgress(progress, total)
//Where progress represents the bytes of motion data received, total represents the
total number of bytes of motion data that need to be synchronized
```

- Set motion data callback

```
CallBackUtils.setSportCallBack(SportCallBack)
```

- Process motion data callback results

```
SportCallBack.onDevSportInfo(DevSportInfoBean)
//App stores or uploads the DevSportInfoBean sports data to the server
```

TIP **BERRY's data synchronization process and interface callback are consistent with APRICOT, BERRY's synchronization speed is improved, and the meaning of progress callback is different**

6. Weather

6.1. APRICOT Weather Process

- After successfully connecting to the device, the App obtains weather source data and updates the watch weather display according to the App definition
- Send the weather data for today + the next N days, N is usually equal to 3, depending on the project requirements

```
/**
* Send daily weather information
* @param WeatherDayBean weather information by day
*/
ControlBleTools.getInstance().sendWeatherDailyForecast(WeatherDayBean,ParsingStateMana
ger.SendCmdStateListener)
```

- Send the weather for the next N hours, N is usually equal to 96, depending on the project requirements

```
/**
* Send weather information for the next hour
* @param WeatherPerHourBean hourly weather information
*/
ControlBleTools.getInstance().sendWeatherPreHour(WeatherPerHourBean,ParsingStateManage
r.SendCmdStateListener)
```

- Send air pressure data

```
/**
* Send air pressure
* @param pressure air pressure data
*/
ControlBleTools.getInstance().sendPressureByWeather(pressure,ParsingStateManager.SendC
mdStateListener)
```

- Listen for device requests to update weather callback

```
CallBackUtils.setWeatherCallBack(WeatherCallBack)
```

- Process device request to update weather callback

```
WeatherCallBack.onRequestWeather()
//Update future weather + hourly weather + air pressure again
```

6.2. BERRY Weather Process

- After successfully connecting to the device, the App obtains weather source data and updates the watch weather display according to the App definition
- Send the latest weather

```
/**
* Send the latest weather
*
* @param LatestWeatherBean latest weather
*/
ControlBleTools.getInstance().sendBerryLatestWeather(BerryLatestWeatherBean,ParsingSta
teManager.SendCmdStateListener)
```

```
public class BerryLatestWeatherBean implements Serializable {

    private BerryWeatherIdBean id;
    private int weather;
    /**
     * Temperature 25
     */
    private BerryWeatherKeyValueBean temperature;
    /**
     * humidity %
     */
    private BerryWeatherKeyValueBean humidity;
    /**
     * Wind force level 0-12
     */
    private BerryWeatherKeyValueBean windSpeed;
     /**
     * Wind deg 0-360
     */
    private BerryWeatherKeyValueBean windDeg;
    /**
     * Sun protection index UV intensity
     */
    private BerryWeatherKeyValueBean uvindex;
    /**
     * Air quality Excellent aqi >=0 && aqi<= 50 Good aqi >50 && aqi<= 100 Slightly
polluted aqi >100
     */
    private BerryWeatherKeyValueBean aqi;
    /**
     * Warning information
     */
    private List<WeatherAlertsListBean> alertsList;
    /**
     *  Atmospheric pressure
     */
    private float pressure;
}

public static class WeatherAlertsListBean {
    /**
     * Warning ID
     */
    private String id;
    /**
     * Warning type String Example: "strong wind"
     */
    private String type;
    /**
     * Warning level String Example: "blue"
     */
    private String level;
    /**
     * Warning title String Example: "Benxi City Gale Blue Warning"
     */
    private String title;
    /**
     * Warning details String Example: "Benxi City Strong Wind Blue Warning Benxi City
Strong Wind Blue Warning Benxi City Strong Wind Blue Warning Benxi City Strong Wind
Blue Warning"
     */
    private String detail;
}
```

- Send the weather data for today + the next N days, N is usually equal to 3, depending on the project requirements

```
/**
* Send future weather - day
*
* @param BerryForecastWeatherBean Future weather - day
*/
ControlBleTools.getInstance().sendBerryDailyForecastWeather(BerryForecastWeatherBean,P
arsingStateManager.SendCmdStateListener)
```

```
public class BerryForecastWeatherBean implements Serializable {
    private BerryWeatherIdBean id;
    public List<WeatherData> data;
}

public class BerryWeatherIdBean implements Serializable {
    /**
     * Millisecond timestamp
     */
    private long pubTime;
    /**
     * City Name
     */
    private String cityName;
    /**
     * Positioning name
     */
    private String locationName;
    /**
     * The location_key field is required when supporting multi-city weather
     * */
    private String locationKey;
    /**
     *Whether the current location of the city, the device supports multiple cities
weather, used to determine whether the current location of the city weather
     * */
    private boolean isCurrentLocation;
}

public static class WeatherData implements Serializable{
        /**
         * Air quality Excellent aqi >=0 && aqi<= 50 Good aqi >50 && aqi<= 100
Slightly polluted aqi >100
         */
        private BerryWeatherKeyValueBean api;
        /**
         * Start weather id - End weather id
         */
        private BerryWeatherRangeValueBean weather;
        /**
         * Minimum temperature - Maximum temperature
         */
        private BerryWeatherRangeValueBean temperature;
        /**
         * Temperature Units
         */
        private String temperatureUnit;
        /**
```

```
         * Sunrise and sunset timestamp in seconds
         */
        private BerryWeatherSunRiseSetBean sunRiseSet;
        /**
         * Wind force level 0-12
         */
        private BerryWeatherKeyValueBean windSpeed;
         /**
         * Wind deg 0-360
         */
        private BerryWeatherKeyValueBean windDeg;
}
```

- Send the weather for the next N hours, N is at most 24

```
/**
* Send future weather - hourly
*
* @param BerryForecastWeatherBean Future weather - hour
*/
ControlBleTools.getInstance().sendBerryHourlyForecastWeather(BerryForecastWeatherBean,
ParsingStateManager.SendCmdStateListener)
```

- Send air pressure data

```
/**
* Send air pressure
* @param pressure air pressure data
*/
ControlBleTools.getInstance().sendBerryPressureByWeather(pressure,ParsingStateManager.
SendCmdStateListener)
```

- Listen for device requests to update weather callback

```
8.CallBackUtils.setWeatherCallBack(WeatherCallBack)
```

- Process device request to update weather callback

```
WeatherCallBack.onRequestWeather()
//Update the latest weather + future weather + hourly weather + air pressure again
```

TIP

BERRY Weather Add the latest weather interface, which can be understood as the latest weather data at the current moment,
The overall weather conditions for a certain day in the next N days. BERRY Weather data structure changes, please refer to DEMO assignment for

details

7. Notice

7.1. APRICOT Notification Process

- Successfully reconnect to the device, the App monitors system notifications and third-party application notifications
- Receive a third-party notification, decide whether to send it based on the App switch control, and send it to call the App notification interface

```
/**
* Send app notifications
*
* @param appName application name
* @param pageName application package name
* @param title Notification title
* @param text notification content
* @param tickerText prompt text
* APP notification: Notification title: 50 Chinese characters long, maximum 150 bytes;
Notification content: 200 Chinese characters long, maximum 600 bytes
*/
ControlBleTools.getInstance().sendAppNotification(appName, pageName, title, text,
tickerText,ParsingStateManager.SendCmdStateListener)
```

- Receive a system call, missed call or SMS, decide whether to send it according to the App switch control, and send it to call the system notification interface

```
/**
* Send system notification
*
* @param type 0 incoming call 1 missed call 2 text message
* @param phoneNumber phone number
* @param contactsInfo contact nickname
* @param messageText (SMS reminder): content body 200 Chinese character string length,
maximum 600 bytes
*/
ControlBleTools.getInstance().sendSystemNotification(type, phoneNumber, contactsInfo,
messageText,ParsingStateManager.SendCmdStateListener)
```

7.2. BERRY Notification Process

- Successfully reconnect to the device, the App monitors system notifications and third-party application notifications
- Receive a third-party notification, decide whether to send it based on the App switch control,

and send it to call the App notification interface

```
/**
* Send app notification
*
* @param key aggregate notification message unique value
* @param appName App name
* @param pageName App package name
* @param title Notification title
* @param text Notification content
* @param tickerText Prompt text
* APP notification: Notification title 50 character string length, maximum 150 bytes;
Notification content body 200 character string length, maximum 600 bytes
*/
ControlBleTools.getInstance().sendAppNotification(key, appName, pageName, title, text,
tickerText,ParsingStateManager.SendCmdStateListener)
```

- Receive a missed call or SMS, decide whether to send it according to the App switch control, and send it to call the system notification interface

```
/**
* Send system notification
*
* @param type  2 SMS messages  !!!Removed 0 incoming calls □1 missed call!!!
* @param key aggregate notification message unique value
* @param sysName system application name
* @param sysPageName system application package name
* @param phoneNumber phone number
* @param contactsInfo contact nickname
* @param messageText (SMS reminder): content text 200 character string length, maximum
600 bytes
*/
ControlBleTools.getInstance().sendSystemNotification(type, key, sysName, sysPageName,
phoneNumber, contactsInfo, messageText,ParsingStateManager.SendCmdStateListener)
```

- App missed call reminder switch control, need to be updated to the device

```
/**
* Set the call-related notification reminder switch
*
* @param isCallOpen Whether to enable incoming call notification reminder
* @param isMissCallOpen Whether to enable missed call notification reminder
* @param listener
*/
public void setBerryCallNotificationSwitch(isCallOpen, isMissCallOpen,
ParsingStateManager.SendCmdStateListener)
```

- App incoming call reminder switch control, need to be updated to the device

```
/**
* Set the incoming call notification reminder switch separately
*
* @param isOpen Whether to enable incoming call notification reminder
* @param listener
*/
ControlBleTools.getInstance().setBerryIncomingCallNotificationSwitch(isOpen,
SendCmdStateListener)
```

- App missed call reminder switch control, need to be updated to the device

```
/**
* Set the missed call notification reminder switch separately
*
* @param isOpen Whether to turn on the missed call notification reminder
* @param listener
*/
ControlBleTools.getInstance().setBerryMissCallNotificationSwitch(isOpen,
SendCmdStateListener)
```

- Handle device request to open application

```
DeviceOpenNotifyAppCallBack.onRequestOpen(pageName)
//App opens the corresponding App according to the package name pageName
```

- App control removes notifications on the device

```
/**
* Remove device notification
* @param packageNames List<String> application package name collection
*/
ControlBleTools.getInstance().removeNotification(packageNames,ParsingStateManager.Send
CmdStateListener)
```

- App settings notification related settings

```
/**
* App settings notification related settings
* @param NotificationSettingsBean notification settings
*/
ControlBleTools.getInstance().setNotificationSettings(NotificationSettingsBean,Parsing
StateManager.SendCmdStateListener)
// NotificationSettingsBean adds the following parameters
///**
```

```
// * Notification only when the phone is locked
// */
//var isOnlyLockedNotify: Boolean = false
///**
// * Wear notification only
// */
//var isOnlyWornNotify: Boolean = false
```

- Set notification screen light settings

```
/**
* Set notification settings parameters
*
* @param bean
* @param listener
*/
ControlBleTools.getInstance().setNotificationSettings(NotificationSettingsBean,
SendCmdStateListener)
/**
* Notification does not light up the screen
*/
NotificationSettingsBean.noticeNotLightUp
/**
* Notification only when the phone is locked
*/
NotificationSettingsBean.isOnlyLockedNotify
/**
* Wear notification only
*/
NotificationSettingsBean.isOnlyWornNotify
```

```
/**
* Get notification settings parameters
*
* @param listener
*/
ControlBleTools.getInstance().getNotificationSettings(SendCmdStateListener)

/**
* Set up monitoring
* */
CallBackUtils.settingMenuCallBack.onNotificationSetting(NotificationSettingsBean)
```

TIP

BERRYSystem notification to remove incoming call reminder type 0, add sysName, sysPageName parameters, Add device request to open application callback,
Add App active removal notification function interface, APRICOT does not

> **support the new interface**

8. Sync Contacts

8.1. APRICOT Contact Synchronization Process

- Successfully reconnect to the device and enter the App contact function
- Call the contact interface

```
/**
* Get contact list
*/
ControlBleTools.getInstance().getContactList(ParsingStateManager.SendCmdStateListener)
```

- Set the listener to get the contact callback

```
CallBackUtils.setContactCallBack(contactCallBack)
```

- Handle the callback of getting contacts

```
ContactCallBack.onContactResult(ContactBean)
```

- Call the contact settings interface

```
/**
* Set up contact list
*
* @param list List<ContactBean> Maximum 10 data
*/
ControlBleTools.getInstance().setContactList(lsit,ParsingStateManager.SendCmdStateList
ener)
```

8.2. BERRY Contact Synchronization Process

- Successfully reconnect to the device and enter the App contact function
- Call the contact interface

```
/**
* Get contact list
*/
ControlBleTools.getInstance().getContactList(ParsingStateManager.SendCmdStateListener)
```

- Set the listener to get the contact callback

```
CallBackUtils.setContactCallBack(contactCallBack)
```

- Handle the callback of getting contacts

```
ContactCallBack.onContactResult(ContactBean)
```

- Call the contact settings interface

```
/**
* Set up contact list
*
* @param list List<ContactBean> Maximum 10 data
*/
ControlBleTools.getInstance().setContactList(lsit,ParsingStateManager.SendCmdStateList
ener)
```

- App sets the monitoring device to request the contact nickname callback

```
CallBackUtils.setBerryDevReqContactCallBack(BerryDevReqContactCallBack)
```

- Handle the callback of the device requesting to obtain the contact nickname

```
BerryDevReqContactCallBack.onDeviceRequestContact(phoneNumber)
//App obtains the system local contact nickname through phoneNumber and calls the send
contact nickname interface

/**
* Update device request contact information
*
* @param name contact nickname
* @param phoneNumber contact number
*/
ControlBleTools.getInstance().updateBerryContactInfo(name, phoneNumber,
ParsingStateManager.SendCmdStateListener)
```

TIP

BERRY sync contactsAdd a callback for the device to actively request to obtain the contact nickname. After receiving the callback, the App needs to obtain the local contact nickname information. If successful, it needs to be sent to the device

9. Get device firmware log

9.1. APRICOT Obtain device firmware log process

- Call to obtain firmware log interface

```
/**
* Apply for device firmware log data upload
*/
ControlBleTools.getInstance().getFirmwareLog(ParsingStateManager.SendCmdStateListener)
```

- Set firmware log callback

```
CallBackUtils.setFirmwareLogStateCallBack(FirmwareLogStateCallBack)
```

- Handle firmware log callback

```
FirmwareLogStateCallBack.onFirmwareLogState(state)
//Processing status FirmwareLogState 0: Start uploading (first packet) 1: Uploading 2:
End data uploading (last packet)

FirmwareLogStateCallBack.onFirmwareLogFilePath(filePath)
//Process the file path filePath
```

9.2. BERRY Get device firmware log process

- Set log related callbacks

```
CallBackUtils.setBerryFirmwareLogCallBack(BerryFirmwareLogCallBack)
```

- Get the type log file status

```
/**
* Request to obtain log file status
*
* @param type @see BerryFirmwareLogCallBack.LogFileType
* @param optionalUserId
* @param optionalDeviceType
* @param optionalPhoneType
*
* @see BerryFirmwareLogCallBack#onLogFileStatus(LogFileStatusBean)
*/
ControlBleTools.getInstance().requestLogFileStatusByBerry(type, optionalUserId,
optionalPhoneType, optionalAppVer, optionalDeviceType,
ParsingStateManager.SendCmdStateListener)
```

- Process callback

```
BerryFirmwareLogCallBack.onLogFileStatus(LogFileStatusBean)
//Judge LogFileStatusBean.fileSize != 0, then proceed to the next step
```

- Apply for firmware to start uploading log

```
/**
* Request to upload or stop uploading log files
*
* @param isStart isStart pass true to start, pass false to end
* @param type @see BerryFirmwareLogCallBack.LogFileType
* @param size is obtained by
BerryFirmwareLogCallBack.onLogFileStatus(LogFileStatusBean)
*/
ControlBleTools.getInstance().requestUploadLogFileByBerry(isStart, type, size,
ParsingStateManager.SendCmdStateListener)
```

- Process upload result callback

```
BerryFirmwareLogCallBack.onLogFileUploadStatus(DeviceFileUploadStatusBean)
//Judge whether DeviceFileUploadStatusBean.isSuccessful is uploaded successfully
```

- After the device returns the file, the App needs to send the application firmware log to complete (the purpose is to make the device delete the local log file record and will not send it next time)

```
/**
* Request to upload or stop uploading log files
*
* @param isStart isStart pass true to start, pass false to end
* @param type @see BerryFirmwareLogCallBack.LogFileType
* @param size is obtained by
BerryFirmwareLogCallBack.onLogFileStatus(LogFileStatusBean)
*/
ControlBleTools.getInstance().requestUploadLogFileByBerry(isStart, type, size,
ParsingStateManager.SendCmdStateListener)
```

- Process file log path callback

```
BerryFirmwareLogCallBack.onLogFilePath(path)
```

- Process device request App to obtain firmware log

```
BerryFirmwareLogCallBack.onDeviceRequestAppGetLog()
// After receiving the callback, call DIMENSION_LOG to start and end the transmission
log, and DUMP_LOG to start and end the transmission log.
```

TIP **BERRY obtains firmware log Change interface name, parameters, and process**

10. Large file transfer

10.1. APRICOT large file transfer process

- Query large file transfer status

```
/**
* Get the status of the device sending large files
*
* @param isForce whether to force update
* @param version version number
* @param md5 md5
* @param ParsingStateManager.SendCmdStateListener callback
*/
ControlBleTools.getInstance().getDeviceLargeFileState(isForce, version, md5,
DeviceLargeFileStatusListener)
```

- Handle the transfer status callback

```
DeviceLargeFileStatusListener.onSuccess(statusValue, statusName)
//If statusValue == DeviceLargeFileStatusListener.PrepareStatus.READY.state, large
files can be sent. Otherwise, large files cannot be sent.
```

- Sending large files

```
/**
* Start uploading large file data
*
* @param type type BleCommonAttributes.UPLOAD_BIG_DATA_*
* @param fileByte file
* @param resumable whether to support breakpoint resumable
* @param ParsingStateManager.SendCmdStateListener callback listener
*/
ControlBleTools.getInstance().startUploadBigData(type,fileByte,
resumable,UploadBigDataListener)
```

- Handle the callback of sending files

```
UploadBigDataListener.onProgress(curPiece, dataPackTotalPieceLength)
//Process the transfer progress

UploadBigDataListener.onTimeout(msg)
//Handle transmission timeout or failure

UploadBigDataListener.onSuccess()
//Processing successful transmission
```

10.2. BERRY large file transfer process

- Query large file transfer status

```
/**
* Get the status of the device sending large files
*
* @param fileBytes file must
* @param fileType The file type must be BleCommonAttributes.UPLOAD_BIG_DATA_*
* @param deviceType device type required
* @param firmwareVersion firmware version (must be passed when type is ota)
* @param ParsingStateManager.SendCmdStateListener
*/
ControlBleTools.getInstance().getDeviceLargeFileStateByBerry(fileBytes, fileType,
deviceType, firmwareVersion, DeviceLargeFileStatusListener)
```

- Handle the transfer status callback

```
DeviceLargeFileStatusListener.onSuccess(statusValue, statusName)
//If statusValue == DeviceLargeFileStatusListener.PrepareStatus.READY.state, large
files can be sent. Otherwise, large files cannot be sent.
```

- Sending large files

```
/**
* Start uploading large file data
*
* @param type type BleCommonAttributes.UPLOAD_BIG_DATA_*
* @param fileByte file
* @param ParsingStateManager.SendCmdStateListener callback listener
*/
ControlBleTools.getInstance().startUploadBigDataByBerry(type,fileByte,
UploadBigDataListener)
```

- Handle the callback of sending files

```
UploadBigDataListener.onProgress(curPiece, dataPackTotalPieceLength)
//Process the transfer progress

UploadBigDataListener.onTimeout(msg)
//Handle transmission timeout or failure

UploadBigDataListener.onSuccess()
//Processing successful transmission
```

> **TIP** **BERRY large file transferChange the interface name and parameters, the rest is the same as APRICOT**

11. Watch Face

11.1. APRICOT dial process

- Online cloud watch face
  - Check the status of watch face file sending

```
/**
* Get the status of sending watch face file
*
* @param watch_face_id Watch face ID
* @param fileSize file size
* @param isReplace whether to replace
* @param ParsingStateManager.SendCmdStateListener callback listener
*/
ControlBleTools.getInstance().getDeviceWatchFace(watch_face_id, fileSize, isReplace,
DeviceWatchFaceFileStatusListener)
```

- Handle dial file sending status callback

```
DeviceWatchFaceFileStatusListener.onSuccess(statusValue, statusName)
//If statusValue == DeviceWatchFaceFileStatusListener.PrepareStatus.READY.getState(),
then sending the watch face file is allowed, otherwise it is not allowed
```

- Calling the interface for sending large files

```
/**
* Start uploading large file data
*
* @param type type BleCommonAttributes.UPLOAD_BIG_DATA_*
* @param fileByte file
* @param resumable whether to support breakpoint resumable
* @param ParsingStateManager.SendCmdStateListener callback listener
*/
ControlBleTools.getInstance().startUploadBigData(type,fileByte,
resumable,UploadBigDataListener)
```

- Handle the callback of sending files

```
UploadBigDataListener.onProgress(curPiece, dataPackTotalPieceLength)
//Process the transfer progress

UploadBigDataListener.onTimeout(msg)
//Handle transmission timeout or failure

UploadBigDataListener.onSuccess()
//Processing successful transmission
```

- Album watch face
    - Get the renderings

```
/**
* Get the renderings
*
* @param src dial bin byte[] resource
* @param r color R value
* @param g color G value
* @param b color B value
* @param bgBmp background Bitmap
* @param textBmp text Bitmap
* @param callBack callback
*/
ControlBleTools.getInstance().myCustomClockUtils(src, r, g, b, bgBmp, textBmp,
EffectCallBack)
```

- Get text overlay effect

```
/**
* Get text image
*
* @param text_bitmp text Bitmap
* @param color_R color R value
* @param color_G color G value
* @param color_B color B value
*/
ControlBleTools.getInstance().newTextBitmap(text_bitmp, color_R, color_G, color_B)
```

- Get the album watch face transfer file

```
/**
* Get the watch face file
*
* @param src dial bin byte[] resource
* @param r color R value
* @param g color G value
* @param b color B value
* @param bgBmp background Bitmap
* @param textBmp text Bitmap
* @param callBack callback
* @param isPositive dial direction positive
*/
ControlBleTools.getInstance().newCustomClockDialData(src, r, g, b, bgBmp, textBmp,
DialDataCallBack, isPositive)
```

- Inquiry about the status of the dial file sending

```
/**
* Get the status of sending watch face file
*
* @param watch_face_id Watch face ID
* @param fileSize file size
* @param isReplace whether to replace
* @param ParsingStateManager.SendCmdStateListener callback listener
*/
ControlBleTools.getInstance().getDeviceWatchFace(watch_face_id, fileSize, isReplace,
DeviceWatchFaceFileStatusListener)
```

- Handle dial file sending status callback

```
DeviceWatchFaceFileStatusListener.onSuccess(statusValue, statusName)
//If statusValue == DeviceWatchFaceFileStatusListener.PrepareStatus.READY.getState(),
then sending the watch face file is allowed, otherwise it is not allowed
```

- Calling the interface for sending large files

```
/**
* Start uploading large file data
*
* @param type type BleCommonAttributes.UPLOAD_BIG_DATA_*
* @param fileByte file
* @param resumable whether to support breakpoint resumable
* @param ParsingStateManager.SendCmdStateListener callback listener
*/
ControlBleTools.getInstance().startUploadBigData(type,fileByte,
resumable,UploadBigDataListener)
```

- Handle the callback of sending files

```
UploadBigDataListener.onProgress(curPiece, dataPackTotalPieceLength)
//Process the transfer progress

UploadBigDataListener.onTimeout(msg)
//Handle transmission timeout or failure

UploadBigDataListener.onSuccess()
//Processing successful transmission
```

- Set installation success callback

```
CallBackUtils.setWatchFaceInstallCallBack(WatchFaceInstallCallBack)
```

- Watch face management
  - Get the watch face list interface

```
/**
* Get installed watch faces
*/
ControlBleTools.getInstance().getWatchFaceList(ParsingStateManager.SendCmdStateListene
r)
```

- Set callback to get watch face list

```
CallBackUtils.setWatchFaceListCallBack(WatchFaceListCallBack)
```

- Set the watch face to the current one

```
/**
* Set the current watch face
*
* @param id dial id
*/
ControlBleTools.getInstance().setDeviceWatchFromId(id,ParsingStateManager.SendCmdState
Listener)
```

- Delete the current watch face

```
/**
```

```
* Delete watch face
*
* @param id dial id
*/
ControlBleTools.getInstance().deleteDeviceWatchFromId(id,ParsingStateManager.SendCmdSt
ateListener)
```

- Set the watch face deletion or set the watch face result callback

```
CallBackUtils.setWatchFaceCallBack(WatchFaceCallBack)

WatchFaceCallBack.setWatchFace(isSet)
//isSet == true, App or watch is set successfully

WatchFaceCallBack.removeWatchFace(isRemoce)
//isRemoce == true, app or watch deleted successfully
```

11.2. BERRY watch face process

- Install the dial and modify the style
  - Check the status of watch face file sending

```
/**
* Get the status before sending the dial
*
* @param watchFaceId watch face iD
* @param fileSize dial style "1" / "2" ...
* @param albumBitmap album background image cloud dial pass null
*/
ControlBleTools.getInstance().getWatchFaceStatusByBerry(watchFaceId, style, fileSize,
albumBitmap,ParsingStateManager.SendCmdStateListener)
```

- Set the dial file status callback

```
CallBackUtils.setBerryWatchFaceStatusCallBack(BerryWatchFaceStatusCallBack)
```

- Process file status callback result

```
BerryWatchFaceStatusCallBack.onPrepareStatus(BerryWatchFaceStatusReplyBean)
//If bean.statusValue==BerryWatchFaceStatusCallBack.PrepareStatus.READY.getState() is
used, the next step is allowed, otherwise it is not allowed.
//If it is an album dial, you need to process the rounded corner value
bean.screenRadius, cut the background image into rounded corners and send it to the
device
```

- Query large file transfer status

```
/**
* Get the status of the device sending large files
*
* @param fileBytes file must
* @param fileType The file type must be BleCommonAttributes.UPLOAD_BIG_DATA_*
* @param deviceType device type required
* @param firmwareVersion firmware version (must be passed when type is ota)
* @param ParsingStateManager.SendCmdStateListener
*/
ControlBleTools.getInstance().getDeviceLargeFileStateByBerry(fileBytes, fileType,
deviceType, firmwareVersion, DeviceLargeFileStatusListener)
```

- Handle transfer status callback

```
DeviceLargeFileStatusListener.onSuccess(statusValue, statusName)
//If statusValue == DeviceLargeFileStatusListener.PrepareStatus.READY.state, large
files can be sent. Otherwise, large files cannot be sent.
```

- Send watch face file

```
/**
* Send watch face file
*
* @param isAlbum Cloud watch face passes the value false, album passes the value true
* @param fileByte value dial BIN file
* @param background background Bitmao cloud dial value passed null
* @param requestBean requestBean value (id dial id, isSetCurrent is true, style style
value background convention 1, 2, 3...)
* @param listener
*/
ControlBleTools.getInstance().startUploadDialBigDataByBerry(isAlbum, fileByte,
background, BerryAlbumWatchFaceEditRequestBean, BerryDialUploadListener)
```

- Handle the callback of sending files

```
//Process the transfer progress
BerryDialUploadListener.onProgress(curPiece, dataPackTotalPieceLength)



//Handle transmission timeout or failure
BerryDialUploadListener.onTimeout(msg)

//Processing successful transmission, refer to BerryAlbumDialUploadListener.SucCode
BerryDialUploadListener.onSuccess(Code)
```

- Set installation success callback

```
CallBackUtils.setWatchFaceInstallCallBack(WatchFaceInstallCallBack)
```

- If you need to modify the dial style and album background separately, you need to transfer the bin file in advance. You can directly call the dial sending interface startUploadDialBigDataByBerry without querying the dial status and large file status. To modify the style, assign BerryAlbumWatchFaceEditRequestBean. To modify the album dial background, assign background.
    - Watch face management
- Get the watch face list interface

```
/**
* Get installed watch faces
*/
ControlBleTools.getInstance().getWatchFaceList(ParsingStateManager.SendCmdStateListene
r)
```

- Set callback to get watch face list

```
CallBackUtils.setWatchFaceListCallBack(WatchFaceListCallBack)

//Processing watch face list results List<WatchFaceListBean>
WatchFaceListCallBack.onResponse(list)

class WatchFaceListBean implements Serializable {
/**
 * Dial ID
 */
 val id:String? = null
/**
 * Is it current
 */
 val isCurrent = false
/**
 * Can it be removed?
 */
 val isRemove = false
/**
 * Dial Name    Newly added by BERRY, not available in APRICOT
 */
 val name:String? = null
/**
 * style    Newly added by BERRY, not available in APRICOT
 */
 val style:String? = null
/**
```

```
 * Image formats supported by the photo watch face  Newly added by BERRY, not
available in APRICOT
 * @see com.zhapp.ble.callback.WatchFaceListCallBack.ImageFormat
 */
 val supportImageFormat = 0
/**
 * Album background dial id Newly added by BERRY, not available in APRICOT
 */
 val backgroundImage:String? = null
 /**
 * Dial Version Newly added by BERRY, not available in APRICOT
 */
 val versionCode:Long = 0
}
```

- Set the watch face to the current one

```
/**
* Set the current watch face
*
* @param id dial id
*/
ControlBleTools.getInstance().setDeviceWatchFromId(id,ParsingStateManager.SendCmdState
Listener)
```

- Delete the current watch face

```
/**
* Delete watch face
*
* @param id dial id
*/
ControlBleTools.getInstance().deleteDeviceWatchFromId(id,ParsingStateManager.SendCmdSt
ateListener)
```

- Set the watch face deletion or set the watch face result callback

```
CallBackUtils.setWatchFaceCallBack(WatchFaceCallBack)

WatchFaceCallBack.setWatchFace(isSet)
//isSet == true, App or watch is set successfully

WatchFaceCallBack.removeWatchFace(isRemove)
//isRemove == true, app or watch deleted successfully
```

TIP BERRY watch face management is consistent with APRICOT, Transfer watch face

> interface callback update, Cloud watch face album watch face related interfaces merged