Reeman 3128 SDK development interface documentation

version:20200410

Instructions for use

- It is only applicable to the development of REEMAN 3128 platform
- 3128 platform equipped with Android 5.1 system development
- Recommended Android Studio
- Development specification follows Google Android standard

`

SDK Preliminary use

1. Create a new Android Studio program, and import Reeman3128.xxxxxxxx.jar into the libs folder, and execute the compilation item under build.gradle to use
2. Register the following permissions in the AndroidManifest.xml file

```
    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission
android:name="android.permission.READ_EXTERNAL_STORAGE"  />
    <uses-permission
android:name="android.permission.WRITE_EXTERNAL_STORAGE"  />
    <uses-permission
android:name="android.permission.ACCESS_NETWORK_STATE"  />
```

1. SDK initialization, sample code is as follows

```
    /**
     * Get ConnectServer object, mContext value must be Application
     */
    ConnectServer cs = ConnectServer.getInstance(mContext);
```

2. SDK registration ROS callback monitoring, please cooperate with reeman ROS string protocol description file to start

```
    /**
     * ROS interface
     */
    private OnROSListener mRosListener = new OnROSListener()
        { @Override
        public void onResult(String data)
            { Log.d("ros onResult", data);
```

```
            /**
             *data is the return value of the ROS protocol communication
             *callback interface, refer to the Reeman ROS serial protocol
             *description document
             */
        }
    };
    /**
     * Register ROS monitor
     */
```

3. SDK logs off ROS monitoring callback and exits

```
/**
    * Set ROS monitor to null
    */
  cs.registerROSListener(null);
  /**
    * Release the ConnectServer object
    */
  cs.release();
```

4. SDK initialization code reference

```
public class RobotHardSdk
  { private static RobotHardSdk
  sdk; private Application
  mContext; private ConnectServer
  cs;
  private RobotHardSdk(Application application)
      { mContext = application;
      cs = ConnectServer.getInstance(mContext);
      registerRos();
  }
  /**
    * Create SDK instance
    *
    * @param application
    */
  public static void CreateInstance(Application application)
      { if (sdk == null) {
          sdk = new RobotHardSdk(application);
      }
  }
  /**
    * Obtain SDK instance
    *
    * @return
    */
  public static RobotHardSdk getInstance()
      { if (sdk != null) {
          return sdk;
```

```
        } else {
            return null;
        }
    }

    public void release() {
        cs.registerROSListener(null);
        if (cs != null)
            cs.release();
        if (sdk != null)
            sdk = null;
    }

    public void registerRos() {
        if (cs == null)
            return;
        cs.registerROSListener(mRosListener);
    }
    /**
     * ROS interface
     */
    private OnROSListener mRosListener = new OnROSListener()
        { @Override
        public void onResult(String data)
            { Log.d("ros onResult", data);
        }
    };
}
```

System broadcast events

- Emergency stop switch broadcast

```
//Action
AUTOCHARGE_ERROR_DOCKNOTFOUND
//Broadcast description
Emergency stop switch press and status report
//value
 int stopState = intent.getIntExtra("SCRAM_STATE", -1);
// -1     Default value, no emergency stop status report
// 1      Emergency stop state is unplugged and open
// 0      The emergency stop status is press to close
```

- Charging point not found broadcast

```
//Action
AUTOCHARGE_ERROR_DOCKNOTFOUND
//Broadcast instructions
Auto-charge docking with charging pile, report the status of not finding the charging pile
//value
nothing
```

- Failed to connect to charging station broadcast

```
//Action
AUTOCHARGE_ERROR_DOCKINGFAILURE
//Broadcast description
Automatic charging docking charging point, reporting the failure of connecting charging point
//value
nothing
```

- Broadcast use reference code

```
RobotReceiver robotReceiver;
//initialization
private void initReceiver() {
      robotReceiver = RobotReceiver.getInstance();
      IntentFilter filter = new IntentFilter();
      filter.addAction("REEMAN_BROADCAST_SCRAMSTATE");
      filter.addAction("AUTOCHARGE_ERROR_DOCKNOTFOUND");
      filter.addAction("AUTOCHARGE_ERROR_DOCKINGFAILURE");
      filter.addAction(Intent.ACTION_BATTERY_CHANGED);
      registerReceiver(robotReceiver, filter);
  }
  //Logout
  private void unReceiver() {
      if (robotReceiver != null)
          { unregisterReceiver(robotReceiver);
      }
  }

  public class RobotReceiver extends BroadcastReceiver  {
     private static RobotReceiver instance;
     public static RobotReceiver getInstance()  {
      if (instance == null) {
          instance = new RobotReceiver();
          }
        return instance;
     }

     private RobotReceiver() {
      }
```

```
        @Override
       public void onReceive(Context context, Intent intent)
              { String action = intent.getAction();
             if ("REEMAN_BROADCAST_SCRAMSTATE".equals(action)) {
                  int stopState = intent.getIntExtra("SCRAM_STATE", -1);
              } else if (Intent.ACTION_BATTERY_CHANGED.equals(action))
                  { int level =
intent.getIntExtra(BatteryManager.EXTRA_LEVEL, 0);
                  int mPlugType =
intent.getIntExtra(BatteryManager.EXTRA_PLUGGED, 0);
              } else if ("AUTOCHARGE_ERROR_DOCKNOTFOUND".equals(action))
                  { Log.d("POWER", "DOCKNOTFOUND");
              } else if ("AUTOCHARGE_ERROR_DOCKINGFAILURE".equals(action))
{
                  Log.d("POWER", "DOCKINGFAILURE");
              }
          }
      }
```

SDK function usage instructions

- ROS instruction set sending

```
    /**
      *The instruction set refers to the Reeman ROS serial protocol
      *description document档
      */
    RobotActionProvider.getInstance().sendRosCom("Instruction
    Set");
```

- Get ROS location value interface(Only online access is supported)

```
RobotActionProvider.getInstance().getPoints(new PointCallback() {
        @Override
        public void getMapPoints(Map<String, String> map) {
            //Detailed location data String is the location name
            /**
              *The map key is the location name, and the value is
              *the coordinate value (x, y, degree value)
              */


        @Override
        public void getListPoints(List<String> list)  {
            //Location data String is the location name
        }


    });
```

- Control the robot forward

```
//param Is the forward distance in cm ，speed Speed value transmission 0
RobotActionProvider.getInstance().moveFront(param,speed);
//Successful execution, ROS Callback value move:done:16
```

- Control the robot back

```
//param Is the forward distance in cm ·speed Speed value transmission 0
RobotActionProvider.getInstance().moveBack(param,speed);
//Successful execution, ROS Callback value move:done:16
```

- Control the robot to turn left

```
//param Is the forward distance in cm ·speed Speed value transmission 0
RobotActionProvider.getInstance().moveLeft(param,speed);
//Successful execution, ROS Callback value move:done:17
```

- Control the robot to turn right

```
//param Is the forward distance in cm ·speed Speed value transmission 0
RobotActionProvider.getInstance().moveRight(param,speed);
//Successful execution, ROS Callback value move:done:18
```

- Control the machine to stop moving

```
//Only effective in manually controlled movements
RobotActionProvider.getInstance().stopMove();
```

- Get machine unique ID

```
// return RobotID
RobotActionProvider.getInstance().getRobotID();
```

- Get current Android system WiFi connection

```
//key for SSID,value for pwd
Map wifiMap = RobotActionProvider.getInstance().getWifiPassword();
```

- Get the current emergency stop status

```
//Reference broadcast report value
RobotActionProvider.getInstance().getScramState();
```

- Restart all (including navigation host)

```
RobotActionProvider.getInstance().rebootAll();
```

- Shutdown

```
RobotActionProvider.getInstance().shutDown();
```

- Whether to force full screen

```
// default false, full screen as true
RobotActionProvider.getInstance().setFullScreen(false);
```

-