NLS-MT65 (4G)
SDK Handbook

Revision History

Version	Description	Date
V1.0.0	Initial release.	October 13, 2015

Table of Contents

About This Manual

This manual is applicable to NLS-MT65 (4G) portable data collectors (hereinafter referred to as **"the MT65"** or **"the terminal"**).

Development Environment

All APIs are built based on standard Android broadcast mechanism, so there is no need for additional SDKs. The MT65 application development environment is the same as Android application development environment.

Obtain Product Model

To get the product model, use android.os.Build.MODEL.

Barcode Scanner

Scan Barcode

To activate the MT65 to scan barcode, application should send the following broadcast to the system.

- Broadcast: nlscan.action.SCANNER_TRIG
- Extra scan timeout parameter: SCAN_TIMEOUT (value: int, 1-9; default value: 6; unit: second)
- Extra scan type parameter: SCAN_TYPE (value: 1–single barcode, 2–twin barcode; default value: 1)

  Note: Twin barcode is not supported currently.

Example 1:

```
Intent intent = new Intent ("nlscan.action.SCANNER_TRIG");
mContext.sendBroadcast(intent);
```

Example 2:

```
Intent intent = new Intent ("nlscan.action.SCANNER_TRIG");
intent.putExtra("SCAN_TIMEOUT", 4);// SCAN_TIMEOUT value: int, 1-9; unit: second
intent.putExtra("SCAN_TYPE ", 2);// SCAN_TYPE: twin barcode
mContext.sendBroadcast(intent);
```

Note: When a scan and decode session is in progress, sending the broadcast above will stop the

ongoing session. When scanning barcode by pressing the Scan key, it is processed at the bottom layer, thus application does not need to listen for Scan KeyPress event or send the broadcast.

Get Barcode Data

There are three ways to get barcode data:

1. Fill in EditText directly: Output scanned data at the current cursor position in EditText.

2. Simulate keystroke: Output scanned data to keyboard buffer to simulate keyboard input and get the data at the current cursor position in TextBox.

3. Output via API: Application acquires scanned data by registering a broadcast receiver and listening for specific broadcast intents.

Broadcast: nlscan.action.SCANNER_RESULT
Extra scan result 1 parameter: SCAN_BARCODE1
Extra scan result 2 parameter: SCAN_BARCODE2
Extra scan state parameter: SCAN_STATE (value: fail or ok)
Parameter type: String

Example:

Register broadcast receiver:

```
mFilter= newIntentFilter("nlscan.action.SCANNER_RESULT");
mContext.registerReceiver(mReceiver, mFilter);
```

Unregister broadcast receiver:

```
mContext.unregisterReceiver(mReceiver);
```

Get barcode data:

```
mReceiver= newBroadcastReceiver() {
        @Override
publicvoidonReceive(Context context, Intent intent) {
            finalString scanResult_1=intent.getStringExtra("SCAN_BARCODE1");
            finalString scanResult_2=intent.getStringExtra("SCAN_BARCODE2");
finalString scanStatus=intent.getStringExtra("EXTRA_SCAN_STATE");
                if("ok".equals(scanStatus)){
                //Success
                }else{
                //Failure, e.g. operation timed out
                }
    }
    };
```

Change the Scanner Settings

Application can set one or more scanner parameters, such as enable/disable scanner, by sending to the system the broadcast ACTION_BAR_SCANCFG which can contain up to 3 parameters.

Parameter	Type	Description (* indicates default)
EXTRA_SCAN_POWER	INT	Value = 0 Disable scanner = 1 Enable scanner* Note: When scanner is enabled, it will take some time to initialize during which all scan requests will be ignored.
EXTRA_TRIG_MODE	INT	Value = 0 Level mode = 1 Continuous mode = 2 Pulse mode*
EXTRA_SCAN_MODE	INT	Value = 1 Fill in EditText directly* = 2 Simulate keystroke = 3 Output via API
EXTRA_SCAN_AUTOENT	INT	Value = 0 Do not add a line feed* = 1 Add a line feed
EXTRA_SCAN_NOTY_SND	INT	Value = 0 Sound notification off = 1 Sound notification on*
EXTRA_SCAN_NOTY_VIB	INT	Value = 0 Vibration notification off* = 1 Vibration notification on
EXTRA_SCAN_NOTY_LED	INT	Value = 0 LED notification off = 1 LED notification on*

Example 1: Disable scanner

```
Intent intent = new Intent ("ACTION_BAR_SCANCFG");
intent.putExtra("EXTRA_SCAN_POWER", 0);
mContext.sendBroadcast(intent);
```

Example 2: Output via API, add a line feed

```
Intent intent = new Intent ("ACTION_BAR_SCANCFG");
intent.putExtra("EXTRA_SCAN_MODE", 3);
intent.putExtra("EXTRA_SCAN_AUTOENT", 1);
mContext.sendBroadcast(intent);
```

Reserved Keys

The MT65 provides 4 reserved keys: F1, F2, F3 and F4. Application can define the functions of these 4

keys as per actual needs.

Example 1: Process the KeyDown event of reserved key

```
public boolean onKeyDown(intkeyCode, KeyEvent event) {
switch (keyCode)
        {
case KeyEvent.KEYCODE_F1:
showInfo("F1 KeyDown\n");
break;
case KeyEvent.KEYCODE_F2:
showInfo("F2 KeyDown\n");
break;
case KeyEvent.KEYCODE_VOLUME_DOWN:
showInfo("F3 KeyDown\n");
break;
case KeyEvent.KEYCODE_VOLUME_UP:
showInfo("F4 KeyDown\n");
break;
        }
```

Example 2: Process the KeyUp event of reserved key

```
public boolean onKeyUp(intkeyCode, KeyEvent event) {
switch (keyCode)
        {
case KeyEvent.KEYCODE_F1:
showInfo("F1 KeyUp\n");
break;
case KeyEvent.KEYCODE_F2:
showInfo("F2 KeyUp\n");
break;
case KeyEvent.KEYCODE_VOLUME_DOWN:
showInfo("F3 KeyUp\n");
break;
case KeyEvent.KEYCODE_VOLUME_UP:
showInfo("F4 KeyUp\n");
break;
        }
return super.onKeyDown(keyCode, event);
    }
```

PSAM Card

Note: Only the MT65 equipped with PSAM card slot supports this feature.

PSAM applications need to call shared library APIs and contain the header file sam_gprot.h (See Appendix 1). The shared library has been integrated in the firmware. The relevant APIs are described as follows.

Open Device

intOpenCard(unsigned long *fd, intslotno);
Parameter: [in]intslotno, slot number (If you pass in 0, then it will automatically adapt to the first open card slot)
[out]unsigned long * fd (Return the device handle status)
Return value: 0 indicates success, non-zero value indicates error

Close Device

intCloseCard(unsigned long fd);
Parameter: [in]unsigned long fd (Pass in the device handle you want to close)
Return value: 0 indicates success, non-zero value indicates error

Reset Device

intResetCard(unsigned long fd, unsigned char *atr,int *atrLen);
Parameter: [in]unsigned long fd (Pass in the device handle you want to close)
[out]unsigned char *atr (Return the device resetting information)
[in/out]int *atrLen (Return the length of device resetting information)
Return value: 0 indicates success, non-zero value indicates error

Send Command

intCardApdu(unsigned long fd, unsigned char* apdu, intapduLength, unsigned char* response,int* respLength);
Parameter: [in]unsigned long fd (Pass in the device handle)
[in]unsigned char *apdu (apdu command you want to send)
[in]intapduLength (Length of apdu command you want to send)
[out]unsigned char*response (Return the data)
[in/out]int* respLength (Return the data length)
Return value: 0 indicates success, non-zero value indicates error

Note: This API does not automatically execute GET RESPONSE command (i.e. it does not automatically send a GET RESPONSE command such as 00c0.

Check Card

intCheckCard(unsigned long fd);
Parameter: [in]unsigned long fd (Pass in the device handle you want to check)
Return value: 0 indicates success, non-zero value indicates error

Development Example

```
#include<sam_gprot.h>
void *SamApiHandle = NULL;
int (*poweron)(char *, int);
int (*poweroff)(void);
int (*reset)(char *, int);
int (*release)(void);
int (*sendapdu)(char* apdu, intapduLength, char **ret_buf);

void initSamApi(void) {
    if (SamApiHandle == NULL) {
SamApiHandle = dlopen("libsam.so", RTLD_GLOBAL);
if (!SamApiHandle) {
printf("load libsam.so error : %s ", dlerror());
}
}
}

void deinitSamApi(void) {
if (SamApiHandle != NULL) {
dlclose(SamApiHandle);
SamApiHandle = NULL;
}
}

int OpenCardInternal(char *atr, intlen) {
const char *error;
int ret = -12345;
    poweron = dlsym(SamApiHandle, "OpenCardInternal");
```

```
if ((error = dlerror()) == NULL) {
ret = poweron(atr, len);
} else {
printf("OpenCardInternal error:%s", error);
}
return ret;
}

int CloseCardInternal(void) {
const char * error;
int ret = -12345;
poweroff = dlsym(SamApiHandle, "CloseCardInternal");

if ((error = dlerror()) == NULL) {
ret = poweroff();
} else {
printf("CloseCardInternal error:%s", error);
}
return ret;
}

int ResetCardInternal(char *atr, intlen) {
const char * error;
int ret = -12345;
      reset = dlsym(SamApiHandle, "ResetCardInternal");

if ((error = dlerror()) == NULL) {
ret = reset(atr, len);
} else {
printf("ResetCardInternal error:%s", error);
}
return ret;
}

int ReleaseCardInternal(void) {
const char * error;
int ret = -12345;
      release = dlsym(SamApiHandle, "ReleaseCardInternal");

if ((error = dlerror()) == NULL) {
```

```
ret = release();
      } else {
printf("ReleaseCardInternal error:%s", error);
      }
return ret;
}

int CardApduTTLInternal(char* apdu, intapduLength, char **ret_buf) {
      const char * error;
int ret = -12345;
sendapdu = dlsym(SamApiHandle, "CardApduTTLInternal");

if ((error = dlerror()) == NULL) {
ret = sendapdu(apdu, apduLength, ret_buf);
      } else {
printf("CardApduTTLInternal error:%s", error);
      }
return ret;
}
```

Other APIs

Expand the Status Bar

To set the status bar to be expandable/not expandable, application should send to the system the broadcast nlscan.action.STATUSBAR_SWITCH_STATE with the value of Extra parameter ENABLE set to be true/false.

Example: Set the status bar to be not expandable

```
Intent intent = new Intent("nlscan.action.STATUSBAR_SWITCH_STATE");
intent.putExtra("ENABLE", false);
context.sendBroadcast(intent);
```

Press the Home Key to Switch to Desktop

To enable/disable the feature of switching to desktop by pressing the Home key, application should send to the system the broadcast nlscan.action.HOMEKEY_SWITCH_STATE with the value of Extra parameter ENABLE set to be true/false.

Example: Disable the feature of switching to desktop by pressing the Home key

```
Intent intent = new Intent("nlscan.action.HOMEKEY_SWITCH_STATE");
intent.putExtra("ENABLE", false);
context.sendBroadcast(intent);
```

Set the System Time

To set the system time, application should send to the system the broadcast nlscan.action.SET_TIME with the value of Extra parameter TIME_MS set to be a string represented as the number of millisecond.

Example:

```
public long getTimeMillis(){
    Calendar c = Calendar.getInstance();
    c.set(2016, 0, 1, 0,0,0);
    return c.getTimeInMillis();
}
Intent it = new Intent("nlscan.action.SET_TIME");
long mills = getTimeMillis();
it.putExtra("TIME_MS", String.valueOf(mills));
mContext.sendBroadcast(it);
```

Appendix 1: Header File sam_gprot.h for PSAM Card Application

```
#ifndef __SAM_GPROT_H_
#define __SAM_GPROT_H_

#ifdef __cplusplus
extern "C"{
#endif

/*1.
Function: Open device
Parameter: [in]intslotno, slot number    (If you pass in 0, then it will automatically adapt to the first open card slot)

[out]unsigned long * fd    (Return the device handle status)
Return value: 0 indicates success, non-zero value indicates error
*/
int OpenCard(unsigned long *fd, intslotno);

/*2.
Function: Close device
Parameter: [in]unsigned long fd    (Pass in the device handle you want to close)
Return value: 0 indicates success, non-zero value indicates error
*/
int CloseCard(unsigned long fd);

/*3.
Function: Reset device
Parameter: [in]unsigned long fd    (Pass in the device handle you want to close)
                [out]unsigned char *atr    (Return the device resetting information)
                [in/out]int *atrLen    (Return the length of device resetting information)
Return value: 0 indicates success, non-zero value indicates error
*/
int ResetCard(unsigned long fd, unsigned char *atr,int *atrLen);

/*4.
Function: Send command
Parameter: [in]unsigned long fd    (Pass in the device handle)
              [in]unsigned char *apdu    (apdu command you want to send)
              [in]intapduLength    (Length of apdu command you want to send)
```

[out]unsigned char*response (Return the data)
[in/out]int* respLength (Return the data length)
Return value: 0 indicates success, non-zero value indicates error
Note: This API does not automatically execute GET RESPONSE command (i.e. it does not automatically send a GET RESPONSE command such as 00c0).
*/
int CardApdu(unsigned long fd, unsigned char* apdu, intapduLength, unsigned char* response,int* respLength);

/*5.

Function: Check card
Parameter: [in]unsigned long fd (Pass in the device handle you want to check)
Return value: 0 indicates success, non-zero value indicates error
*/
int CheckCard(unsigned long fd);

#ifdef __cplusplus
}
#endif

#endif

Headquarters
Fujian Newland Auto-ID Tech. Co., Ltd.
3F, Building A, No.1, Rujiang West Rd., Mawei, Fuzhou, Fujian, China 350015
TEL: +86 - (0) 591-83979222
FAX: +86 - (0) 591-83979208
E-mail: marketing@nlscan.com
WEB：www.nlscan.com

Newland Europe BV
Rolweg 25, 4104 AV Culemborg, The Netherlands
TEL: +31 (0) 345 87 00 33
FAX: +31 (0) 345 87 00 39
Email: info@newland-id.com
WEB: www.newland-id.com
Tech Support: tech-support@newland-id.com

Newland North America Inc.
Address: 46559 Fremont Blvd., Fremont, CA 94538, USA
TEL: 510 490 3888
Fax: 510 490 3887
Email: info@newlandna.com
WEB: www.newlandna.com

Newland Taiwan Inc.
7F-6, No. 268, Liancheng Rd., Jhonghe Dist. 235, New Taipei City, Taiwan
TEL: +886 2 7731 5388
FAX: +886 2 7731 5389
Email: info@newland-id.com.tw
WEB: www.newland-id.com.tw