Prolin Application Development Manual

PAX Computer Technology（Shenzhen）Co.,Ltd.

1 Overview

As an auxiliary document of “Prolin API Programming Guide”, this document introduces Prolin-2.x platform application development for Prolin operating system developers.

2 Prolin Major Version

Prolin OS includes the following 7 major versions:

Major Version	Models
Prolin-2.4	D200, S300, S800, S900, S920
Prolin-phoenix-2.5	P^X5, P^X7
Prolin-cygnus-2.6	Q80, Q90, Q30, QR55, D220, IM300
Prolin-pelican-2.7	SP200, Q20, QR55, B920, D190, IM500, IM700
Prolin-peng-2.8	QR68(4G), Q92
Prolin-albatross-2.9	D199, Q92, D230
Nuxiums-3.1	Q25, Q58, D270, D205, D195, SP320, IM15

These 7 major versions are not compatible with each other, but they share some similarities and differences.

2.1 Similarity

- **API and SDK**

  All 7 versions use the same set of encapsulated APIs and Prolin SDK tool.

- **Serial Port Soft Link**

  Since different models may have different serial port interfaces, the

Since different models may assign different physical serial ports for the same logical port (such as PORT_COM1), the soft link is created for the physical serial ports so that applications can use and recognize them easily.
All 7 versions have created the soft link, such as ttycom1 corresponds to PORT_COM1, ttycom2 corresponds to PORT_COM2 and ttypinpad corresponds to PORT_PINPAD. You can check the existence of PINPAD port through the “/dev/ttypinpad” node.
If an application needs to use the POSIX interface, it is recommended to operate “/dev/ttycom1” directly. Calling *open* function to open “/dev/ttycom1” has the equivalent effect as calling `OsPortOpen()` to open PORT_COM1.

2.2 Difference

- **TM Interface Default Setting for Power Management**
  All 7 versions enter the screensaver state if you has not interacted with the terminal for more than 60 seconds.
  - For Prolin-2.4, the automatic sleep function is disabled by default.
  - For Prolin-phoenix-2.5, the TM interface does not contain the automatic sleep option.
  - For other 5 versions, except that Q80 and the models without built-in battery will never enter the sleep state, other models will enter sleep state after being in the screensaver state for 2 seconds.
- **Video Playing**
  - For Prolin-2.4, not support video playing.
  - For other 6 versions, support video playing. For more information, refer to Prolin Player Programming Guide(V1.0.4) or LVGL Programming Guide in the "Video Playback" section.
- **Poweroff Confirmation**

- For Prolin-2.4, no shutdown confirmation interface by default. Except for S300, any other models will be powered off by holding down the power key for a few seconds.
- For Prolin-phoenix-2.5, no shutdown confirmation interface by default. The P^X5 and P^X7 PINPAD models do not have a shutdown button.
- For other 5 versions, shutdown confirmation interface will display after holding down the power key for a few seconds. Press the "Confirm" button to shut down, or the "Cancel" button to cancel the shutdown (Only Q20 and D190 with Prolin-pelican-2.7 have power keys).

- **Multi-touch**
  - For Prolin-2.4, Prolin-pelican-2.7, Nuxiums-3.1, not support multi-touch.
  - For other 4 versions, support multi-touch.

- **Gesture Recognition**
  - For Prolin-cygnus-2.6, Prolin-pelican-2.7, support gesture recognition. For more information, refer to Prolin XUI Interface(V2.1.7)
  - For other 5 versions, not support gesture recognition.

- **Screen Rotation**
  - For Prolin-cygnus-2.6, Prolin-pelican-2.7, Nuxiums-3.1, support screen rotation. The interfaces of TM, PED and application support 90°, 180° and 270° rotation but the LOGO interface doesn't support rotation. Screen rotation can be implemented through TM or by setting the keyword "persist.sys.ui.rotate". For more information, refer to the chapter " Screen Rotation".
  - For other 4 versions, not support screen rotation.

- **Compilation Toolchain**
  When compiling applications through the SDK, you need to choose the correct toolchain.
  - For Prolin-albatross-2.9, Nuxiums-3.1, use gcc-8.3.0.
  - For other 5 versions, use gcc-4.6.3.

3 Prolin Application

The application needs to refer to “osal.h” and link to “libosal.so”, where “osal.h” includes all the interfaces mentioned in the Prolin API Programming Guide(V2.7.2) .

Prolin also provides its own GUI tool called XUI. When using XUI to program, “xui.h” needs to be included and “libxui.so” needs to be linked as well.

4 Package Application

Prolin SDK can be used to package applications automatically. And the application can also be packaged manually. There are two methods shown as below:

Method I: Use the packaging command “genaip” provided by Prolin in Linux environment

1. Copy “genaip” (the application package command provided by Prolin) to a certain directory of PATH for the convenience of the following access.

2. Make an application directory, such as “demo_app”;

3. Copy the executable files, dynamic libraries, and other resource files to the “demo_app” directory, and a soft link isn’t supported.

4. Write “appinfo” files.

- “appinfo” is a standard “ini” file and supports UTF-8 encoding, but it does not support annotation.
- “appinfo” is used for the application loader to get the correct application information. The application installation package must have “appinfo” file.

- The fields supported by “appinfo” are shown in the below table. Except for the keywords in the list, you can self-define fields and keys whose values are within a row. The application can perform other functions by self-defined fields and keys.

Field/Key	Necessity	Description
[app]	Y	Application field
id=demo_app	Y	Unique ID of application (demo_app), C language string variable. The main application’s ID is MAINAPP.
name=demo app	Y	Application name, which is used for displaying and can support UTF-8 encoding.
bin=iPaxMobile_sig	Y	Name of bin program(iPaxMobile_sig)
artwork=	N	Path for storing app icon file
desc=ADemo application	N	Description of application and it only occupies one line.
vender=PAX	N	Application vender
version=1.0.1	N	Application version number

Content of appinfo:

```
[app]
id=demo_app
name=demoapp
bin=iPaxMobile_sig
artwork=
sdesc=A Demo application
vender=PAX
version=1.0.1
/* after installation, the application has read-only permission for the “appinfo” file.*/
```

5. Package through the following commands.

```
/*the directory of application package running "cd" command */
cddemo_app
/*package “demo_app.aip” and self-define the name.*/
genaip -I pkginfo -o demo_app.aip
/*any compression tool that supports standard zip is supported, such as winrar.*/
```

Method II: Package the application without the command “genaip” in Linux environment

1. Create a new folder “test”, in which there are “demo_app” folder, “Makefile” file and “pkginfo” file. “demo_app” folder is used for storing the application files. The executable file and “appinfo” file must be in the root directory of the “demo_app” folder. Other files (which are optional) needed by the application such as other executable files, dynamic libraries and resource files can be stored under “bin”, lib and “res” directory. The writing rule of “appinfo” file is the same as the above method.

2. Write “Makefile” file and “pkginfo” file. “Makefile” file specifies the application name. “pkginfo” file describes all the resource files of the generated AIP package. “appinfo” and “bin” files are required, “lib” and “res” files are optional and iPaxMobile_sig is the executable file.

- “Makefile” file

```
AIP=test.aip
all: aip
aip :
cddemo_app&amp;amp;amp;zip ../$(AIP) -r -MM -@ < ../pkginfo
clean:
	-rm $(AIP)
```

- “pkginfo” file

```
appinfo
iPaxMobile_sig
bin/
lib/
res/
```

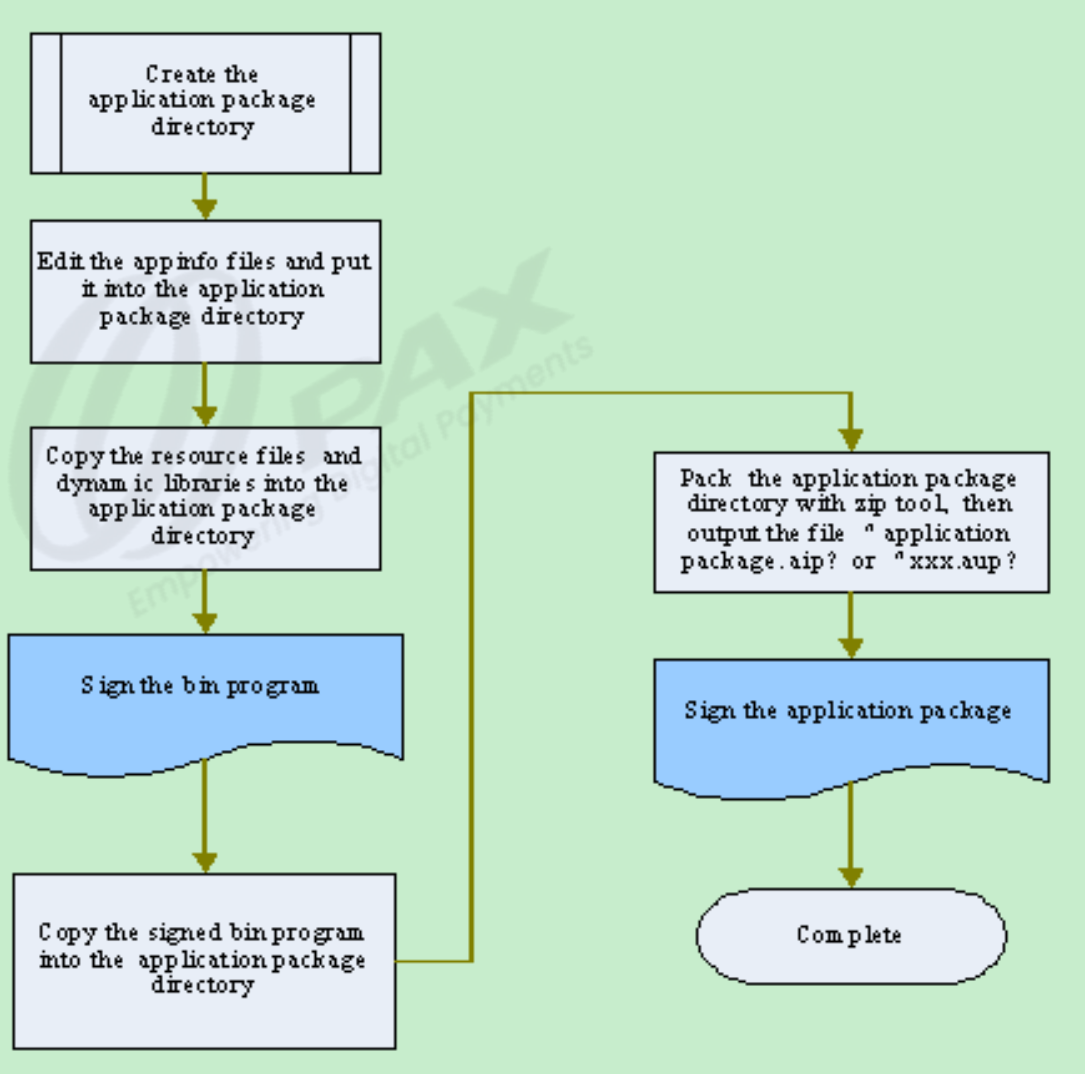
3. Run the command “make” in the root directory of the “test” folder. If you need to re-package the application, run the command “make clean” and “make” in sequence.

4. The packaged AIP file named “test.aip” will be under the root directory of the “test” folder.

5 Sign Application

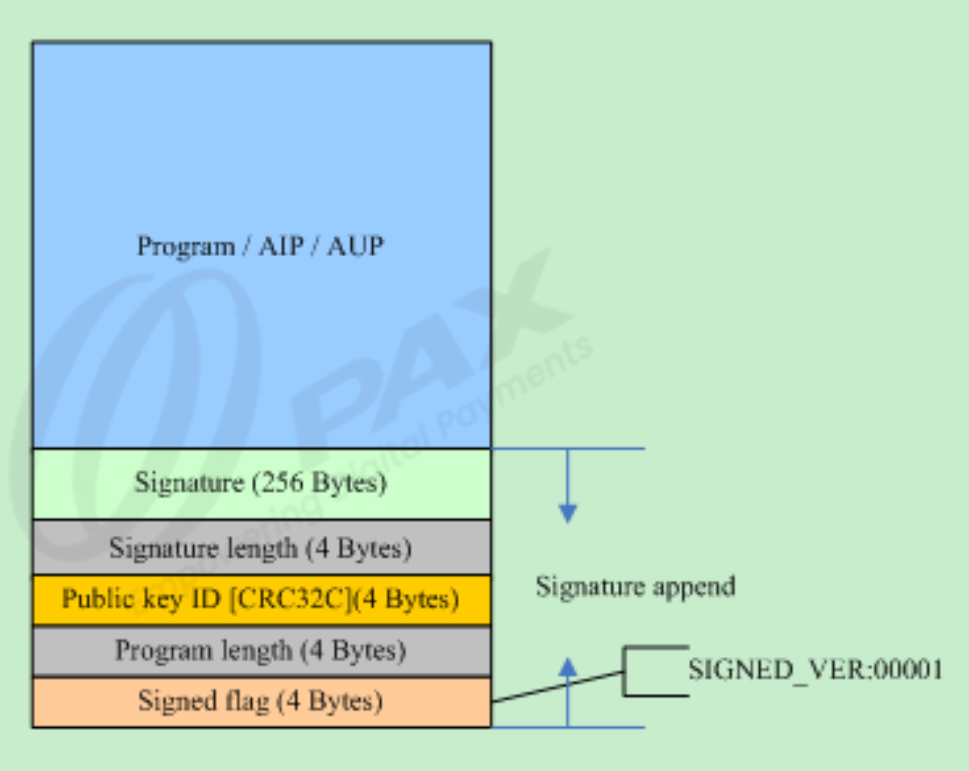
5.1 Signature Process

To guarantee the safety of the installation and execution, the application program and the application package need to be signed. The detailed process is as follows:



5.2 Signature Format

The signature information is appended at the end of the application package. The format is shown as below:



6 Run Application

The system can install multiple applications, including one MAINAPP and several sub-applications, and all the sub-applications are managed by MAINAPP, including installation, uninstallation and deployment. The sub-application has the same independent group permissions. The number of installed applications depends on the remaining space of the system. Sub-applications cannot access each other's resources and dynamic libraries.

When starting the system, MAINAPP will be started if it exists, otherwise, TM will get started. After MAINAPP has been installed, you can start MAINAPP by exiting TM, and then use MAINAPP to start other applications.

MAINAPP can run the sub-application by calling `OsRunApp()` shown as below:

- **The code of sub-application:** take `OsBeep()` as an example, the running parameters are the beep type and the duration, and output the standard output information and error information respectively.

```
int main(int argc, char *argv[])
{
    fprintf(stdout, "argv[1]=%s, argv[2]=%s\n", argv[1], argv[2]);
    fprintf(stderr, "do the OsBeep()\n");
    OsBeep(atoi(argv[1]), atoi(argv[2]));
    return 0;
}
```

- The code of MAINAPP.

```
typedef struct {
    char msg[128];
    int flag;
}INFO_STRUCT;
void cb_output(char *appid, char *str, void *data)
{
    INFO_STRUCT *buf = NULL;
    if(NULL == appid || NULL  == str || NULL == data)
    {
        return;
    }
    buf = (INFO_STRUCT *)data;
    printf("CbOut---APP: %s, INFO: %s, msg: %s, flag: %d\n", appid, str,
buf->msg, buf->flag);
    OsLog(LOG_INFO, "CbOut---APP: %s, INFO: %s, msg: %s, flag: %d\n",
appid, str, buf->msg, buf->flag);
}
void cb_error(char *appid, char *str, void *data)
{
    INFO_STRUCT *buf = NULL;
    if(NULL == appid || NULL  == str || NULL == data)
    {
        return;
    }
    buf = (INFO_STRUCT *)data;
    printf("CbErr---APP: %s, ERR: %s, msg: %s, flag: %d\n", appid, str,
buf->msg, buf->flag);
```

```
    OsLog(LOG_ERROR, "CbErr---APP: %s, ERR: %s, msg: %s, flag: %d\n",
appid, str, buf->msg, buf->flag);
}
int main()
{
    int ret;
    char appid[] = "subapp";  //the id of subapplication to be run
    char *para[3];          //the input parameter of subapplication
    INFO_STRUCT info = {0};   //the input parameter of CbOut and CbErr
    OsLogSetTag("runapp_test");
    //set the para for subapp
    para[0] = "1";
    para[1] = "500";
    para[2] = NULL;
    //set the para for callback fuction
    strcpy(info.msg, "msg for cb");
    info.flag = 1;
    ret = OsRunApp(appid, para, &amp;info, cb_output, cb_error);
    OsLog(LOG_INFO, "OsRunApp return %d\n", ret);
    return 0;
}
```

7 Dynamic Library

Dynamic libraries support private and public libraries. The private library is the library in the “lib” folder under the main directory of the application, which can be downloaded together with the application or as an application parameter file; Public library is the library under the “/opt/lib” folder and can only be downloaded as OPT packages.

8 Font Library

Prolin adopts UTF-8 encoding and Truetype vector font.

All fonts are stored in the “/opt/fonts” file folder. The specific number of font libraries depends on the remaining space of the system. The system library is “paxfont.ttf” with “Times New Roman” style by default. It only supports ASCII code and Latin alphabet which includes English, French, etc, but Chinese, Japanese or Korean are not supported.

8.1 Font Encoding

Unicode encoding cannot run in the real environment, it needs to be converted to variable length code (UTF-8). English character length is 1 byte, and Chinese character length is 3 bytes. Specific conversion rules are as follows:

Unicode encoding(hexadecimal)	UTF-8 byte stream (binary)
0000 - 007F	0xxxxxxx
0080 - 07FF	110xxxxx 10xxxxxx
0800 - FFFF	1110xxxx 10xxxxxx 10xxxxxx

It defaults to use GBK encoding in the Chinese version of the Windows system. When writing a program in the simulator, code conversion needs to be implemented. Specific methods are as follows:

1. Convert GB2312, GBK to UTF-8. The conversion can be completed through the following function.

   ```
   int OsCodeConvertUtf8 (const char *Charset, const char *InBuf, char *OutBuf, unsigned int LenOut)
   ```

2. Convert Unicode to UTF-8. `OsCodeConvertUtf8()` does not support this conversion, so you need to write the following codes:

```
int UnicodeToUtf8(const unsigned int Unicode, char *Utf8Code)
{
	int len = -1;
	if(Unicode < 0x7F){
		Utf8Code[0] = (char)Unicode;
		Utf8Code[1] = 0x00;
		len = 1;
	}
	else if(Unicode < 0x7FF){
		Utf8Code[0] = (0xC0 | (Unicode >> 6));
		Utf8Code[1] = (0x80 | (Unicode &amp; 0x3F));
		Utf8Code[2] = 0x00;
		len = 2;
	}
	else if(Unicode < 0xFFFF){
		Utf8Code[0] = (char)(0xE0 | (Unicode >> 12));
		Utf8Code[1] = (char)(0x80 | (Unicode &amp; 0xFFF) >> 6);
		Utf8Code[2] = (char)(0x80 | (Unicode &amp; 0x3F));
		Utf8Code[3] = 0x00;
		len = 3;
	}
	return len;
}
```

3. Convert UTF-8 to Unicode. For the convenience of debugging, sometimes UTF-8 code also needs to be converted to Unicode as follows:

```
unsigned int Utf8ToUnicode(const char *Utf8Code)
{
	int len;
	unsigned int uiCharCode = 0;
	len = strlen(Utf8Code);
	switch (len) {
	case 1:
		if (Utf8Code[0] &amp; 0x80)
			return -1;
		uiCharCode = (int)Utf8Code[0];
```

```
            break;
        case 2:
            if (!(Utf8Code[0] &amp;amp; 0xC0) &amp;amp;&amp;amp; (Utf8Code[1]
&amp;amp; 0x80))
                return -1;
            uiCharCode = ((int)(Utf8Code[0] &amp;amp; 0x1F) << 5) |
              (int)(Utf8Code[1] &amp;amp; 0x3F);
            break;
        case 3:
            if (!(Utf8Code[0] &amp;amp; 0xE0) &amp;amp;&amp;amp; (Utf8Code[1]
&amp;amp; 0x80) &amp;amp;&amp;amp;
              (Utf8Code[2] &amp;amp; 0x80))
                return -1;
            uiCharCode = ((int)(Utf8Code[0] &amp;amp; 0x0F) << 12) |
                    ((int)(Utf8Code[1] &amp;amp; 0x3F) << 6) |
                    (int)(Utf8Code[2] &amp;amp; 0x3F);
            break;
        default:
            return -1;
        }
        return uiCharCode;
}
```

8.2 Vector Font

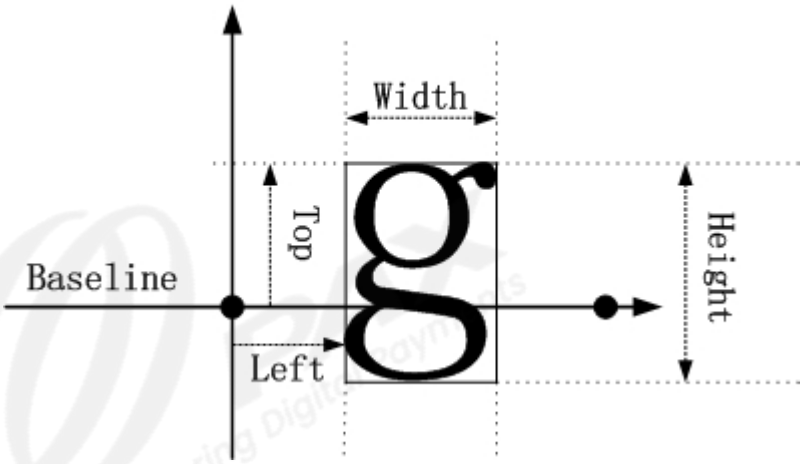
1. **Introduction**

   Glyph of vector font is described by the mathematical curve, it contains key points on Glyph boundary and the derivative information of curve. The font rendering engine will read these vectors and then have certain calculations to render.

2. **Supported by the open-source library**

   Adopt the Freetype - 2.4.8 library with Linux standard.

   a. To store the information of a vector font (as shown in the below figure), the sample code is as follows.




```
typedef struct {
unsigned char  Left;/*Font left offset from the baseline*/
unsigned char  Top;  /*Font top offset from the
baseline*/
unsigned char   Width;      /*Font width*/
unsigned char   Height;       /*Font height*/
unsigned char   Dot[768];      /*Valid font data*/
}FT_DOT;
```

b. To acquire vector fonts and pad them into dot matrix, the sample code is as follows.

```
/* acquire vector fonts and pad them into dot matrix */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <osal.h>
/*display the dot matrix*/
static void
ShowBitMap (unsigned char *DotBuf, int Width, int Height)
{
 int iRow, iCol;
int indexByte;
```

```
 for (iRow = 0; iRow < Height; iRow++){
  printf ("\n");
 for (iCol = 0; iCol < Width; iCol++){
   indexByte = iRow * ((Width + 7) / 8) + iCol / 8;
   if ((DotBuf[indexByte] &amp;amp; (0x80 >> (iCol % 8))))
        printf ("*");
    else
        printf (" ");
}
}
printf ("\n\n");
}
/*pad the acquired dot matrix to the word slot*/
static void
FillDot(FT_DOT FtDot, int Width, int Height, unsigned char *DotBuf, int
Size)
{
int indexByte, startRows, startWidth;
 int iRow, iCol;
 int posByte;
 int BASELINE = Height / 4;
if (FtDot.Top > (Height - BASELINE))        /* eg. hanzi */
    startRows = (Height - FtDot.Height + 1) / 2;
else                                        /* eg. alpha */
startRows = Height - FtDot.Top - BASELINE;
startWidth = (Width - FtDot.Width) / 2;
memset(DotBuf, 0x00, Size);
for (iRow = 0; iRow < FtDot.Height; iRow++) {
for (iCol = 0; iCol < FtDot.Width; iCol++) {
posByte = iRow * ((Width + 7) / 8) + iCol / 8;
if ((FtDot.Dot[posByte] &amp;amp; (0x80 >> (iCol % 8)))) {
indexByte = (startRows + iRow) * ((FtDot.Width + 7) / 8) +
(startWidth + iCol) / 8;
DotBuf[indexByte] |= (0x80 >> ((startWidth + iCol) % 8));
}
}
}
}
```

```
/*acquire vector font dot matrix*/
static int
GetFreetypeDot(const char *Utf8Code, int Width, int Height, FT_DOT
*FtDot)
{
int ret;
int Handle;
Handle = OsOpenFont("simsun.ttc");
if(Handle < 0)
return -1;
ret = OsGetFontDot(Handle, Utf8Code, Width, Height, FtDot);
if(ret < 0)
return -2;
OsCloseFont(Handle);
return 0;
}
int main()
{
FT_DOT FtDot;
int Width = 24;
int Height = 24;
int ret;/* ensure that font is UTF-8 encoding; otherwise it only can
directly fill with internal encoding.*/
char *Utf8Code = "好";
unsigned char DotBuf[512];
ret = GetFreetypeDot(Utf8Code, Width, Height, &amp;FtDot);
if(ret < 0)
return -1;
ShowBitMap(FtDot.Dot, FtDot.Width, FtDot.Height);
ret = FillDot(FtDot, Width, Height, DotBuf, sizeof(DotBuf));
if(ret < 0)
return -2;
ShowBitMap(DotBuf, Width, Height);
}
```

9 OPT Package

Prolin supports the following OPT packages:

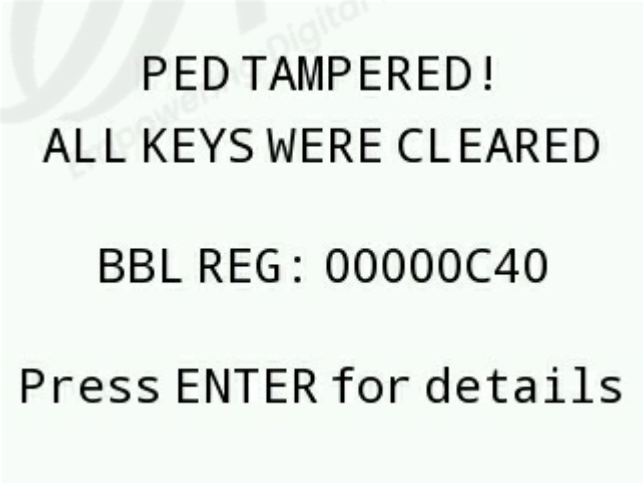
- **Bluetooth Library**: it supports Bluetooth functions and Bluetooth development.
- **Libcurl:** it provides supports for dict, file, ftp, ftps, gopher, http, https, imap, imaps, pop3, pop3s, rtsp, scp, sftp, smb, smbs, smtp, smtps, telnet and tftp communication protocols.
- **Wenquanyi Font:** it is a Chinese font which supports the Chinese display.
- **C++ Package:** the "libstdcpp" installation package which includes libstdc++.so.6 and is used for C++ programming.
- **Two-dimension Code Library:** it is used for encoding the data to the bar code bitmap information. Printing and displaying interfaces can be used to print and display the two-dimension code.
- **Video Player Component** (only PXX series model): it is used for playing video. Only Prolin-phoenix-2.5 and Prolin-cygnus-2.6 systems support playing video, and their corresponding component packages are "player_phoenix" and "player_cygnus". For "player_phoenix", it only supports AVI format video files that have been converted by PAX conversion tool. For "player_cygnus", it can support most type of video formats except rmvb and rm formats. There is an executable file named "player" in the component package, and the application can use the process substitution method (exec) to call the command "player" to implement the video playing function.

10 PED

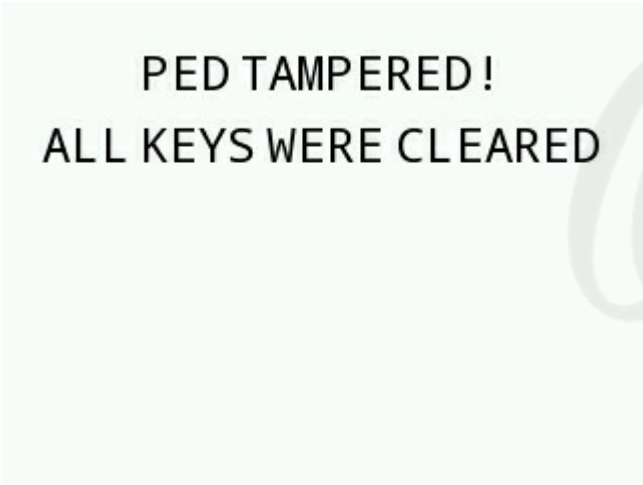
10.1 PED Tamper

Prolin POS tamper includes soft tamper and hard tamper.

- **Hard tamper** is caused by a physical attack, for example, taking the POS terminal apart. The hard tamper interface is as follows:



- **Soft tamper** is caused by the PED key being damaged or lost. For example, when using TermAssist to update Prolin OS, if you selects and executes “Erase data partition”, the data partition will be erased and the PED key will be lost, which causes PED soft tamper. The soft tamper interface is as follows:



10.2 Set PIN Input Background

Call `OsPedSetPinBg()` to use the specified image data or Framebuffer data to set the PIN input background, and the operating instructions are as follows:

- When Mode = 0x01, the Framebuffer data obtained by XUI will be used to set the PIN input background, and the specified steps are listed as below:
  a. XUI creates a canvas that has the same size with the screen as the PIN input background;
  b. Call `XuiCaptureCanvas()` to store PIN background in XuiImg format to the memory.
  c. Call `XuiImgToFrameBuffer()` to convert XuiImg data to Framebuffer data.
  d. Call `OsPedSetPinBg()` to pass the Framebuffer data to PED, and complete the PIN input background setting.

- When Mode = 0x02, the data that is read from image resource files will be used to set PIN input background, and the specified steps are listed as below:
  a. Read image resource files and cache them in memory;
  b. Call `OsPedSetPinBg()` to pass the image data to PED, and complete the PIN input background setting.

- When Mode = 0x00, the settings of Mode = 0x01 and Mode = 0x02 will be cleared. The sample code is as follows:

```
#define SCREEN_SIZE          ...
int SetPinBgFun(void)
{
int iRet, DataLen;
unsigned char DataBuff[SCREEN_SIZE];
unsigned char Mode;
XuiOpen();
OsPedOpen();
Mode = 0x01;
```

```
if (Mode == 0x01){
    /* a full-screen canvas must be created by XUI, and the PIN
background should be designed on this canvas.  */
    ...
    /* after the canvas creation, call XuiCaptureCanvas() to store PIN
background in XuiImg format to the memory, but not needs to call
XuiShowWindow() */
    XuiImg *Img = XuiCaptureCanvas(ScreenWindow,0,0,ScreenWindow-
>width,ScreenWindow->height);
    /* convert XuiImg data to Framebuffer data */
    DataLen = XuiImgToFrameBuffer(Img, DataBuff, sizeof(DataBuff));
    ...
    /* call OsPedSetPinBg() to pass the Framebuffer data to PED, and
complete the PIN input background setting */
    iRet = OsPedSetPinBg(Mode, DataBuff, DataLen);
    ...
    /* input PIN  */
    OsPedGetPinBlock(...);
}
    Mode = 0x02;
if (Mode == 0x02) {
    /* read image resource files and caches them in memory */
    iRet = iReadFile(LOGO_BMP, sizeof(DataBuff), DataBuff);
    ...
    /* call OsPedSetPinBg() to pass the Framebuffer data to PED, and
complete the PIN input background setting. */
    iRet = OsPedSetPinBg(Mode, DataBuff, DataLen);
    ...
    /* input PIN */
    OsPedGetPinBlock(...);
}
XuiClose();
OsPedClose();
return iRet;
}
```

10.3 Customize PIN Input Virtual Keypad

This function applies to Prolin-cygnus-2.6.21 or higher version. You can call `OsPedCustomKeypad()` to customize PIN input virtual keypad in terminals without physical keypads, and terminals with physical keypads do not support this function.

When customizing the virtual keypad, the requirements of keypad icon resource are listed as below:

1. Each key corresponds to an icon, the keypad icon resolution of relevant models is shown in the following table.

Model	Screen orientation	Numeric key (0-9)	Confirm key (enter)	Clear key (clear)	Cancel key (cancel)
D220	horizontal (w*h) vertical (w*h)	121*60 148*60	121*60 148*60	121*60 148*60	-
QR55	horizontal (w*h) vertical (w*h)	- 106*46	- 106*46	- 106*46	- 35*52
Q20	horizontal (w*h) vertical (w*h)	- 79*54	- 79*107	- 79*52	- 79*107
D199	horizontal (w*h) vertical (w*h)	135*72 152*72	135*72 152*72	135*72 152*72	135*72 152*72

2. The naming rule of keypad icons is <mode>-<key value>.<suffix>.

- Mode 0 indicates the virtual keypad in vertical screen mode, mode 1 indicates the virtual keypad in horizontal mode;
- Key values can only be set to 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, enter, clear or cancel;
- The suffix is keypad icon format, and it only supports png format currently. For example, in D220 (vertical screen), the icon which corresponds to the key [1] is named “0-1.png”.

11 GPRS/CDMA

Prolin GPRS module supports CMUX multi-channel function. After calling OsWlInit(), the two paths “/dev/mux0” and “/dev/mux1” will be created automatically. “mux0” is for Ipservice and “mux1” is for sending AT command.

WCDMA module supports serial port connection and USB connection, USB connection supports multiple channels while serial port connection doesn’t.For serial port connection, only in command mode (before calling OsWlLogin() or after calling OsWlLogout( )), can AT command be sent through “/dev/ttyw1”.For USB connection, AT command can be sent at any stage through “/dev/mux1”.

Whether the 3G module supports serial port or USB can be checked by “/dev/ttyw1”.

- If /dev/ttyw1->ttyAMAx(x represents number), it is serial port mode.
- If /dev/ttywl->ttyUSBx, it is USB mode.

12 Route Setting

Prolin adopts Linux standard routing management and it can switch different routes to be the default route. Route setting related interfaces include `OsNetSetRoute()` and `OsNetGetRoute()`.

In standard routing management, one device might have several routes but only

one default route. When the “socket” connection did not bind to the fixed network port, “socket” will firstly pair with routes one by one in the routing table. Once “socket” and route are in the same network segment, then the corresponding route will be used. If “socket” did not pair with any of the routes in the routing table, the default route will be used as a network port.

If the application only needs a single link, after calling related interfaces such as `OsWlLogin()`, `OsNetSetConfig()` and `OsNetStartDhcp()` to establish link successfully, this link will be configured as a default route. If there are several links, the last established link will be set to the default link. `OsNetSetRoute()` will set the specified route in the routing table to the default route. If a specified route did not exist in the routing table, the configuration will fail. In the Debug mode, the command “route” can be used to check the Kernel route.If a route and the other end IP address are located in the same network segment, the connection will use the route set by `OsNetRoute()`.

13 D200/S920 Usage Limits

S920 and D200 must work with a battery. If plugging in the external power supply at the same time, the power must be standard, not USB power. Otherwise POS terminal will become abnormal because computer USB only offers 0.5A current.

When POS terminal’s battery is absent and only has an external power supply, RF, printer, Bluetooth, Wi-Fi or GPRS module can’t be used.When both external power supply and battery exist, if battery shows empty, calling `OsWlLock()`, `OsWifiOpen()`, `OsPrnOpen()` or `OsPiccOpen()` will return ERR_BATTERY_VOLTAGE_TOO_LOW.If the battery shows empty when starting the terminal, the terminal needs to be charged first.

When there is only battery to provide power that is too low, the function `OsCheckBattery()` will return 0, then the system will automatically power off.

14 Debug/Release Version

14.1 Version Comparison

For applications, the biggest distinction between Debug version and Release version is whether the application needs a signature. Debug version doesn't need a signature, while the Release version needs a signature.

Apart from the signature difference, you also need to pay attention to differences listed in the following table:

Function Points	Release	Debug
Busybox	N	Y
/proc	N	Y
Shell	N	Y
Console	N	Y (Permission: MAINAPP)

Because "Shell" has been deleted from Release version, `system()` and `popen()` of Linux POSIX interface cannot be called. According to the above table, it is noted that the applications that run on the Debug version may not work on the Release version after signature.

In addition, when starting the Debug version terminal, the screen will show "!WARNING!" interface as follows, you can press the "Enter" key to skip the interface.

14.2 Version Switching

Prolin-2.4 and Prolin-phoenix-2.5 systems adopt the Boot version of “boot_2.x.x.x” which applies to the non-authorization mechanism and “boot_3.x.x.x” version which applies to the authorization mechanism.

- For non-authorization mechanism models, you can change the configuration file to switch mode between Release and Debug. When the keyword “prolin_debug_level” in the configuration file is equal to 0, it means Release mode; when it is equal to 1, it means Debug mode.
- For authorization mechanism models, the keyword “prolin_debug_level” in the configuration file is no longer valid, that is why an authorization tool is needed when it comes to switching mode. For information on how to use the authorization tool, refer to Authorization Tool Operations Manual.

Prolin-cygnus-2.6 and Prolin-pelican-2.7 systems adopt “boot_4.x.x.x” version and “boot_5.x.x.x” version respectively, and it only supports switching mode through authorization tool.

15 File Copy over Application Layer

File copy in this context means the installed AIP, OPT or PUK file in sending end

(hereafter referred to as A POS) can be copied to the receiving end (hereafter referred to as B POS) through a USB cable or serial port line. And AIP or OPT files in B POS can be uninstalled through A POS.

When copying file, the A POS needs to have XCB client service, therefore the system version requirements for copying file are as follows:

- Prolin-2.4.51 or higher
- Prolin-phoenix-2.5.5 or higher
- Prolin-cygnus-2.6.1 or higher
- Prolin-pelican-2.7.1 or higher

15.1 XCB Command

The file types and the installation/uninstallation commands supported by XCB commands are as follows:

```
/*install or uninstall the application package*/
xcb installer aip <aip-file>
xcb installer uaip <appid>
/*install puk file*/
xcb installer puk<puk-file>
/* install the application upgrade package*/
xcbinstaller aup <aup-file>
/*install or uninstall system optional component
package*/
xcb installer opt <opt-file>
xcb installer uopt <opt-id>
```

15.2 Copy Steps

There are 7 steps in the file copy process, the details are as follows:

1. Set up the physical link.

2. Save the status information of A POS xcbd server, close the local xcbd server.

```
OsRegGetValue("persist.sys.xcb.enable",enablebuf);
OsRegSetValue("persist.sys.xcb.enable", "0");
```

NOTE: MAINAPP permission is needed when copying application files because `OsRegSetValue()` needs MAINAPP permission.

3. Call “xcb connect” in A POS to implement the connection between A POS XCB and B POS xcbd; The sample code is as follows:

```
argv[0] = "xcb";
argv[1] = "connect";
argv[2] = link;   /*if usb cable is being used to copy file, then the link is "com:/dev/ttyhost"; if serial port is being used, the link is "com:/dev/ttyamax" (x value depends on the value of "ro.kernel.console")*/
argv[3] = NULL;
pid = fork();
if(pid < 0)
return -1;
else if(pid == 0){
ret = execvp("xcb", argv);
if(ret)
_exit(127);
}else{
while (waitpid(pid, &amp;status, 0) < 0) {
if (errno != EINTR)
   break;
}
}
return WEXITSTATUS(status);
```

4. Call “xcb installer” in A POS, send testfile from A POS to B POS through physical link, then B POS installs testfile; The sample code is as follows:

```
myargv[0] = "xcb";
myargv[1] = "installer";
myargv[2] = type;   /*the type of installation file, including aip,puk, aup,
etc*/
myargv[3] = filename;    /*the path of file which includes the filename*/
myargv[4] = NULL;
pid = fork();
if(pid < 0)
return -1;
else if(pid == 0){
ret = execvp("xcb", myargv);
if(ret == -1)
_exit(127);
}else{
while (waitpid(pid, &amp;status, 0) < 0) {
if (errno != EINTR)
   break;
}
}
return WEXITSTATUS(status);
```

5. Call “xcb disconnect” in A POS to disconnect the connection between A POS XCB and B POS xcbd. The sample code is as follows:

```
argv[0] = "xcb";
argv[1] = "disconnect";
argv[2] = link;    /*physical link between A POS and B POS
*/
argv[3] = NULL;
pid = fork();
if(pid < 0)
return -1;
else if(pid == 0){
```

```
ret = execvp("xcb", argv);
if(ret)
   _exit(127);
}else{
while (waitpid(pid, &amp;status, 0) < 0) {
   if (errno != EINTR)
      break;
}
}
return WEXITSTATUS(status);
```

6. Call “xcb kill-server” in A POS to close local XCB background server; The sample code is as follows:

```
argv[0] = "xcb";
argv[1] = "kill-server";
argv[2] = NULL;
pid = fork();
if(pid < 0)
return -1;
else if(pid == 0){
ret = execvp("xcb", argv);
if(ret)
_exit(127);
}else{
while (waitpid(pid, &amp;status, 0) < 0) {
if (errno != EINTR)
   break;
}
}
return WEXITSTATUS(status);
```

7. Restore the xcbd server status to the status before file copy in A POS with the following function.

```
OsRegSetValue("persist.sys.xcb.enable", enablebuf);
```

15.3 Notes

1. The way that B POS runs the xcbd server needs to conform to the practical connection mode. For example, when copying files through USB, you need to select USB as XCB service in the TM menu on B POS.
2. Before A POS calls “xcb connect” to set up the connection, save the status of current xcbd server and close the xcbd server. After copying the file successfully, restore the xcbd server status.
3. When copying files through a USB cable, please ensure that A POS is a USB host and B POS is a USB device.
4. When installing or uninstalling files, pass in the right parameters according to the XCB supported command. For example, pass in the parameters “aip” and “*.aip” when installing the application package, and pass in the parameters “uaip” and “appid” when uninstalling the application. Otherwise, it will cause an error.
5. The supported connection of file copy is implemented through USB cable or serial port line. Because of the transmission speed limitation of the serial port line, it is recommended to adopt USB cable to copy files.

16 Change TM Password over Application Layer

In Prolin operating system, you need to enter a fixed password of “123456” to enter the “System Config” interface.

The earlier version of Prolin OS uses AES algorithm to encrypt the plaintext password set by the TM system, and obtains and preserves the password ciphertext with a length of 17 bits in the “persist.sys.tm.pwd” registry. To enhance password safety, for Prolin-2.4.57, Prolin-phoenix-2.5.5, Prolin-cygnus-2.6.1, Prolin-pelican-2.7.1 or higher versions, the plaintext password will be calculated by SHA256 algorithm and it supports a combination of 6-20 digit numbers. The password will be saved as ciphertext of 63 bits and as parameter “persist.sys.tm.pwd” in the registry table.

AES algorithm applies to all Prolin OS versions, while SHA256 algorithm does not apply to Prolin-2.4.56, Prolin-phoenix-2.5.4 or lower versions. That is, the password which is set with OS version equal to or higher than Prolin-phoenix-2.5.5 or Prolin-2.4.57 is rolled back to the OS version equal to or lower than Prolin-phoenix-2.5.4 or Prolin-2.4.56, as a result, the password will not work. You can only reset the parameter “persist.sys.tm.pwd” in the registry table through the application or format the data partition to restore the original password.

16.1 Configuration Method

16.1.1 Configure with TM

If TM password is known, enter “System Config->System Setting->Password” menu to modify TM password.

16.1.2 Configure with Encrypting Algorithm

You can set TM password through algorithm calculation. For more details, refer to the section "Algorithm Implementation" in this chapter.

16.1.3 Copy Password

You can get the ciphertext of password by calling `OsRegGetValue()`, and set password by calling `OsRetSetValue()`. The steps of setting TM password are as follows:

1. Set TM password in TM interface or by SHA256 hash value of plaintext password on A POS.
2. Run A application calling `OsRegGetValue("persist.sys.tm.pwd", value)` on A POS, and get the ciphertext on A POS.
3. Run B application calling `OsRegSetValue("persist.sys.tm.pwd", value)` on B POS. And the password gotten from A POS will be set as a password of B POS. It is noted that B application must be the main application.

16.2 Algorithm Implementation

The following codes implement plaintext password calculation by SHA256 algorithm. When compiling codes, -lcrypt library needs to be linked, header files that need to be included are as follows, and the macro "_GNU_SOURCE" must be defined.

- **Sample code**

```
#define _GNU_SOURCE
#include <stdio.h>
#include <crypt.h>
#include <ctype.h>
#include <time.h>
#include <stdlib.h>
```

- **GetRandomSalt()**

```
int GetRandomSalt(char *buf, int len, int num)
{
    int i=0;
    if(!buf ||len < num+3) {
        return -1;
    }
    buf[0] = '$';
    buf[1] = '5';
    buf[2] = '$';
    for(i=0; i< num; i++) {
        srand(time(NULL)+i);
        buf[i+3] = (char)(((double)rand() / ((double)RAND_MAX + 1) * 10)) +
48;
    }
    buf[num+3] = '$';
    return 0;
}
```

- **TMSysconfigPwdConfig()**

```
int TMSysconfigPwdConfig(char *PlainPwd)
{
    int iRet,len, i;
    char Salt[64] = {0};
    const char *ShaPwd = NULL;
    struct crypt_data Data = {0};
    if (PlainPwd == NULL){
        fprintf(stderr, "ERR: Invalid Plain Password.\n");
        return -1;
    }
    len = strlen(PlainPwd);
    if (len < 6 || len > 20){
        fprintf(stderr, "ERR: Invalid Plain Password Length.\n");
        return -2;
    }
```

```
    for (i=0; i<len; i++){
        if (!isdigit(PlainPwd[i])){
            fprintf(stderr, "ERR: Invalid Plain Password Format.\n");
            return -3;
        }
    }
    memset(Salt, 0, sizeof(Salt));
    GetRandomSalt(Salt, sizeof(Salt), 16);
    Data.initialized = 0;
    ShaPwd = crypt_r(PlainPwd, Salt, &amp;Data);
    OsRegSetValue("persist.sys.tm.pwd", ShaPwd);
    return 0;
}
```

17 Time Zone Setting

17.1 Setting Steps

Only Prolin-2.4.60, Prolin-phoenix-2.5.8, Prolin-cygnux-2.6.1, Prolin-pelican-2.7 or higher versions support time zone and daylight saving time settings. The setup method is as follows:

Method I: Set through MAINAPP

The way to set the time zone and daylight saving time through MAINAPP is to change *persist.sys.timezone* of registry table information. The specific steps are as follows:

1. Ensure that the current time is synchronous with the local time.
2. Call `OsGetTime(ST_TIME \*Time)` to get the current system time (See Prolin API Programming Guide for details);
3. Call `OsRegSetValue(const char *Key, const char *Value)` to set "persist.sys.timezone.tz" to the local time such as CST-8 and CET-1CEST,M3.5.0,M10.5.0/3. For more information on "tz", refer to the Linux

"tz" value. For example, on Ubuntu system, you can check the tz value in /usr/share/zoneinfo directory or refer to the website link http://www.gnu.org/software/libc/manual/html_node/TZ-Variable.html.

4. Call `OsSetTime(ST_TIME \*Time)` to set the time obtained in step 2 to the current system time.
5. After the setting is done, reboot the terminal to ensure that the system time is synchronous with the local time.

Method II: Set through XCB tool

Firstly, you need to follow the steps below to check whether XCB tool supports setting time zone or not. (Here taking Windows 7 as an example):

In Windows control console, firstly using the command "cd" to enter the "tools" directory of installation path of TermAssist tool (taking **C:\Users\wangyf\Desktop\TermAssist3.0.4.29 47\tools** as an example). Then running xcb.exe, if there is prompt information containing "xcb settime <yyyyMMddHHmmss[tz]> -set device time", it indicates XCB tool supports time zone setting.

If XCB tool supports setting the time zone, using the following steps to set:

1. Set up the link between XCB tool and POS terminal (For more information, refer to Prolin-2.x User Guide).
2. Execute the command xcb settime 20160421171000CST-8 to set the system time to 17:10:00, April 21st, 2016 and the system time zone to CST-8.
3. After step 2 has been executed successfully, run the command "xcb gettime" to check whether the setting is successful or not.
4. After setting is done, reboot the POS terminal so that the system time will be synchronous with the local time.

> NOTE
>
> The command format of setting system time is `settime yyyyMMddHHmmss[tz]`, of which, [tz] time zone setting is optional. If only setting time is needed, then the format is `xcb settime yyyyMMddHHmmss`.

17.2 Notes

1. For switching from an OS that does not support time zone setting to a POS terminal that can set the time zone, if the time zone characteristic needs to be used, any one of the above methods needs to be adopted to set time zone information.
2. It is noted that the POS terminal needs to be rebooted after the time zone setting is done so that the system time will be synchronous with the local time, otherwise, it might cause system time disorder.
3. When setting the time zone, ensure that the *tz* value must be correct to change the environment parameter "persist.sys.timezone.tz", otherwise, the system will adopt the UTC value.
4. The daylight saving time is slightly different for different places, therefore, when setting the Prolin time zone, refer to the rule of the local daylight saving time.

18 Left Memory and Flash Space

For Prolin OS, you can call `sysinfo()` to acquire the system remaining memory or `statfs()` to acquire the flash left available space. Both of these two functions are located in Glibc, and their implementation do not rely on /proc file. Examples are as follows:

- **sysinfo() example**

```
//function prototype:
#include <sys/sysinfo.h>
int sysinfo(struct sysinfo *info);
//code segment:
```

```
struct sysinfo St;
memset(&amp;St, 0, sizeof(St));
if(sysinfo(&amp;St) < 0){
perror("sysinfo");
return -1;
}
printf("totalram: %lu MB, freeram: %lu MB\n ", St.totalram>>20,
St.freeram>>20); //the total memory storage of pos terminal and left
available memory space.
```

- **statfs() example**

```
//function prototype:
#include <sys/statfs.h>
int statfs(const char *Path, struct statfs *Buf);
//code segment:
struct statfs DiskInfo;
unsigned long long TotalSize = 0;
unsigned long long AvailableDisk = 0;
if(statfs("/data", &amp;DiskInfo) < 0) {
perror("statfs");
return -1;
}
TotalSize = DiskInfo.f_bsize * DiskInfo.f_blocks;
printf("Total size = %llu MB\n", TotalSize>>20);
AvailableDisk = DiskInfo.f_bavail * DiskInfo.f_bsize;
printf("Disk available size = %llu MB\n", AvailableDisk >>20); // the left
available memory space of flash.
```

19 Nand Flash Lifespan

Frequent write operations on the file system can lead to flash bit flips, causing

read/write errors. To address this issue, a method for querying Nand Flash block erase information is provided. The specific steps are as follows:

1. Read the contents of the `/proc/mounts` file to obtain the UBI device `/dev/ubiN_X` corresponding to the `/data` directory.
2. Extract the UBI device number N from the UBI device name.
3. Read the following files in the `/sys/devices/virtual/ubi/ubiN/` directory to obtain Nand Flash erase information:
   - `stat_max_ec`: Records the erase count of the most frequently erased Nand Flash block.
   - `stat_mean_ec`: Records the average erase count of the Nand Flash block.
   - `stat_sum_ec`: Records the total erase count of the Nand Flash block.

> NOTE
>
> The above erasing information of Nand Flash block is saved in UBI file system. When using TermAssist to erase the data partition, the above information will be cleared.

When developing an application, it is possible to simulate real usage scenarios to test the erase count of Nand Flash within a day or a week. If the erase count is too high, optimizations can be made promptly to avoid the application writing files too frequently, which can lead to bit flips in the Flash memory. This could eventually cause file system errors and terminal tamper.

The maximum erase count for Nand Flash is generally 100,000 times. Assuming the terminal is designed for a lifespan of 5 years, the erase count of Nand Flash per day should not exceed 20,000/365 = 54 times. The actual erase count can be reasonably adjusted based on the terminal's actual usage lifespan.

Below is an example code to test the average erase count of the Nand Flash in a day.

```
int get_erase_count(char *file_name)
```

```
{
 int erase_count = -1, len = 256;
 char line[256] = {0}, path[256] = {0};
 char data_dir[] = "/data", ubi_tag[] = "/dev/ubi";
 char ubi_num;
 FILE *mounts_file = NULL, *ec_file = NULL;

 mounts_file = fopen("/proc/mounts", "r");
 if (!mounts_file) {
  return -1;
 }

 while (!(feof(mounts_file))){
  memset(line, 0, len);
  if (fgets(line, len, mounts_file) == NULL){
   break;/*end of file*/
  }

  if(strstr(line, data_dir)){
   if(strstr(line, ubi_tag)){
    ubi_num = *(strstr(line, ubi_tag) + strlen(ubi_tag));
    snprintf(path, sizeof(path), "/sys/devices/virtual/ubi/ubi%c/%s", ubi_num,
file_name);
    ec_file = fopen(path, "r");
    if (!ec_file){
     break;
    }

    memset(line, 0, len);
    if (fgets(line, len, ec_file) != NULL){
     erase_count = atoi(line);
    }

    fclose(ec_file);
    break;
   }
  }
 }
```

```
 fclose(mounts_file);
 return erase_count;
}

int main(void)
{
 int erase_count = -1, erase_count_bak = -1;

 /*get the average erasing count of Nand Flash block.*/
 erase_count_bak = get_erase_count("stat_mean_ec");
 if (erase_count_bak < 0){
  return -1;
 }

    .....     //after 24h

 erase_count = get_erase_count("stat_mean_ec");
 if (erase_count < 0){
  return -1;
 }

 if((erase_count - erase_count_bak) > 54) {
  OsLog(LOG_ERROR, "Warning!!! Write files too frequently!");
 }

 return 0;
}
```

20 Log Information

Traditional Prolin stored the log information only in SDRAM, and the log information will be lost after the system restarts. The latest Prolin uses “logcat service” to manage the log information, and the log information will be saved in a

specified log file. The system log file can contain information up to 2.5M, and you can check the history log information after rebooting the device.

20.1 Export Steps

The methods of checking and exporting log information are as follows:

1. For terminals with COM port, enter the command “logcat”**in the console** to check log information.

2. Enter the command “logcat” **in XCB telnet** function to check log information.

> NOTE
> Method 1 and method 2 do not support the logcat service. Only the log information generated after the device starts up will be output, and it doesn’t include the history log information, and those two methods only apply to the POS terminal in Debug mode.

3. **TermAssist** exports log information (only applies to the TermAssist version 3.0 or above), for more information, refer to section 3.1.5 in Prolin-2.x User Guide . This method does not support the logcat service. Only the log information generated after the device starts up will be output. If you already has implemented a cleanup log operation through TermAssist, then the exported information will be the log information after cleanup. But TermAssist cleanup implementation has no influence on the system log file, the cleared log information will still be stored in the system log file.

4. **USB disk** exports log information. Enter TM->System Report->Diagnosis menu and export the log information to the USB disk. For more information, refer to the section “System Report” in Prolin-2.x User Guide . The state of logcat service can be checked by this method. If the service is closed, only the log information generated after the device starts up can be exported to USB disk; if the service is

open, the log information will be exported from the system log file directly, and thus the history log information can be exported too.

> NOTE
>
> Both method 3 and method 4 apply to POS terminal in Debug mode and Release mode. Only method 4 supports the logcat service.

For the convenience of distinguishing the history log and new log information, each time when system get started, the related startup log and time stamp will be given to analyze and debug easily, an example is shown as below:

```
I/keyman  (  894): ****************************************
I/keyman  (  894):
I/keyman  (  894): system starting up 16:35:47 in 05.25.2016
I/keyman  (  894):
I/keyman  (  894): ****************************************
I/keyman  (  894): System services starting up
I/keyman  (  894): System services startup done
```

20.2 Logcat Service Settings

Logcat service is closed by default and can be switched in TM or by modifying the registry value. When the logcat service is closed, you can still check and export log information by the traditional method (details refer to the former section). However, the log information will not be stored in the system log file, so it will be lost after the system restarts.

Methods of switching the logcat service are as follows:

1. Switch the logcat service in TM.
   Logcat service can be switched in “TM->System Config->Save Log” menu.

2. Switch the logcat service by modifying the registry value.
   The registry parameter of controlling logcat service is "persist.sys.logcat.enable", you can call `OsRegSetValue(const char *Key, const char *Value)` defined in OSAL to set the parameter value. Setting value to "1" indicates that the logcat service is open; while setting it to "0" indicates that logcat service is closed. Details of API refer to Prolin API Programming Guide(V2.7.2) .

> ⚠ CAUTION
>
> Turning this function on for a long time can seriously affect Flash life, so it is recommended to turn it on during the development and debugging phase, but not in a commercial environment.

21 Cppcheck

Cppcheck is a static code analysis tool for the C and C++ programming languages. It is a versatile tool that can check non-standard code, such as memory leak and handle leak. It only applies to SDK-2.8.8 or above. For more

detailed information about cppcheck, refer to Prolin SDK Operating Guide .

22 Multi-Touch

Prolin OS provides an input subsystem, which is in charge of the keypad and touchscreen devices. It will get the input information reported from keypad or touchscreen drivers through reading the corresponding device node. Of which, the device nodes for the keypad and touchscreen are “/dev/keypad” and “/dev/tp” respectively. The input event conforms to the data structure of struct “input_event” (defined in /usr/include/linux/input.h) shown as follows:

```
struct input_event
{
    struct timeval time;
    __u16 type;
    __u16 code;
    __s32 value;
};
```

For Prolin-2.4 system, the touchscreen device adopts the single touch report mode, while for Prolin-phoenix-2.5 and Prolin-cygnus-2.6 systems, they adopt a multi-touch mode. When the application layer is performing the related UI design with the touchscreen, the touch mode of the touchscreen does not need to be focused if XUI interfaces have been used to program. However, if the XUI interface does not use to program but through the reports of touchscreen device nodes receiving from “input” subsystem, please pay attention to the differences of processing reports between the single touch and multi-touch mode. Sample codes are as follows:

- **Single touch operation sample code**

```
//open node
```

```
int fd = open("/dev/tp", O_RDONLY);
if (fd < 0) {
return;
}
//read data from node
struct input_event ievent;
int ret = read(fd, &amp;ievent, sizeof(ievent));
if (ret != sizeof(ievent)) {
return;
}
switch(ievent.type){
case EV_ABS: {  /*touchscreen input, the value of node is calculated
according to the absolute value of coordinate.*/
 switch (ievent.code) {
   case ABS_X:  /*acquire the x coordinate of event, the corresponding
value is "ievent.value" */
break;
    case ABS_Y:  /*acquire the y coordinate of event, the corresponding
value is "ievent.value"*/
break;
  }
}
break;
……
case EV_SYN: /*the end mark of ievent*/
break;
}
```

- **Multi-touch operation sample code**

```
//open node(same as the above sample code)
//read data from node
struct input_event ievent;
int ret = read(fd, &amp;ievent, sizeof(ievent));
if (ret != sizeof(ievent)) {
return;
```

```
}
switch(ievent.type){
case EV_ABS: {          /*touchscreen input, the value of node is
calculated according to the absolute value of coordinate.*/
switch (ievent.code) {
   case ABS_MT_SLOT:    /* acquire the slot sequence of current ievent*/
break;
   case ABS_MT_POSITION_X:  /* acquire the x coordinate
of event, the corresponding value is "ievent.value"
*/
break;
    case ABS_MT_POSITION_Y:  /* acquire the y coordinate of event, the
corresponding value is "ievent.value"*/
break;
   case ABS_MT_TOO_TYPE:   /* acquire input type, it can be a stylus or
finger*/
break;
case ABS_MT_TRACKING_ID:    /* acquire the only id of touching the
screen*/
break;
}
}
break;
……
case EV_SYN: /**the end mark of ievent*/
break;
}
```

23 File System Mount

Prolin supports mounting USB disk and SD card such type of external file systems, Prolin provides APIs for regular mount and encrypted mount.

23.1 APIs

The APIs of external file system mount and unmount are listed in the following table, for more information on parameters and return values, refer to “Prolin API Programming Guide”.

Function Name	Description
OsMount	Regular mount external file system
OsUmount	Regular unmount external file system
OsCryptFormat	Format encrypted file
OsCryptMount	Mount encrypted file system
OsCryptUmount	Unmount encrypted file system

23.2 Mounting Comparison

`OsMount()` and `OsCryptMount()` are both used to mount the file systems under “/dev/block/“ directory to “/mnt/“. It is worth noticing that the external devices normally have partitions, such as USB-disk, which can support up to 4 partition file systems. `OsMount()` supports mounting the whole file system of the external device, while `OsCryptMount()` must need a certain partition of an external device to mount. The example is as follows:

- Insert a USB-disk, the device node under the “/dev/block/“ directory is sda and the device node naming rule is sda, sdb, sdc, etc. The partition node is sda1 and when there are several partitions, the partition node naming rule is sda1, sda2, sda3, etc. At this time, OsMount() can mount “/dev/block/sda” or “/dev/block/sda1”, but OsCryptMount() can only mount “/dev/block/sda1”.

- Insert a SD card, the device node is mmcblk0 under the “/dev/block/” directory and the device node naming rule is mmcblk0, mmcblk1, mmcblk2, etc. The partition node is mmcblk0p1 and when there are several partitions, the partition node naming rule is mmcblk0p1, mmcblk0p2 and mmcblk0p3, etc. At this time, Osmount() can mount “/dev/block/mmcblk0” or “/dev/block/mmcblk0p1” while OsCryptMount() can only mount “ /dev/block/mmcblk0p1”.

Regular mount and encrypted mount are two independent mounting methods, when programming, it is forbidden to mix them up, which means file system mounted by OsMount() can only be unmounted through OsUmount(). While the encrypted file system mounted by OsCryptMount() can only be unmounted by OsCryptUmount().

24 C++ Programming

For Prolin-2.4.34, Prolin-phoenix-2.5.1, Prolin-cygnus-2.6.1, Prolin-pelican-2.7 or higher version OS, C++ library “libstdc++.so.6” is supported and saved under /lib directory. And it is used for C++ programming.

25 Crash Report

There are at least two ways to get control over and extract data from a crashed application in Linux:

- Add default signal handler to a process, which handles abnormal signals.
- Override /proc/sys/kernel/core_pattern file with a piped command (available since Linux 2.6.19).

Android’s debuggerd implements the first approach. In Prolin, crash report is based on debuggerd and it is called **crashd** (“tombstones”) – since it does not

offer debugging, but only collects crash information.

25.1 Work Principle

Crashd listens to UNIX domain socket /tmp/crashd and waits to be connected. Because TID (thread ID obtained by the gettid(2) system call) is supposed to be written to the socket when crashd is connected, the ptrace(2) system call is used to attach to this TID and read from the process memory.

Supports threaded programs (dumps both information for main process and crashed thread) and name resolution.

libosal library contains a function for program developers: `OsSaveCrashReport()`. Before the program starts, signal handlers must be set up for signals that are emitted on abnormal program behavior (e.g.: SIGABRT, SIGSEGV, etc).An example to set up a custom signal handler is as follows:

```
void OsSaveCrashReport(int signal)
{
debugger_signal_handler(signal);
}
```

25.2 Sample Code

- Call `signal (SIG_XXX, OsSaveCrashReport)` to register OsSaveCrashReport as signal handling function. The sample code is as follows:

```
signal(SIGILL,    OsSaveCrashReport);
signal(SIGABRT,   OsSaveCrashReport);
signal(SIGBUS,    OsSaveCrashReport);
signal(SIGFPE,    OsSaveCrashReport);
signal(SIGSEGV,   OsSaveCrashReport);
```

```
signal(SIGSTKFLT, OsSaveCrashReport);
```

- Call `OsSaveCrashReport (sig)` in the signal processing function to save the crash reports. The sample code is as follows:

```
#include "osal.h"
...
int my_signal_handler(int signal)
{
do_something();
OsSaveCrashReport(signal);
}
```

25.3 Crash Report

Upon the crash, trace information is saved under /data/tombstones directory, named “tombstone_<number>“. By default, 20 reports are saved; if there are more, the oldest report will be overwritten.

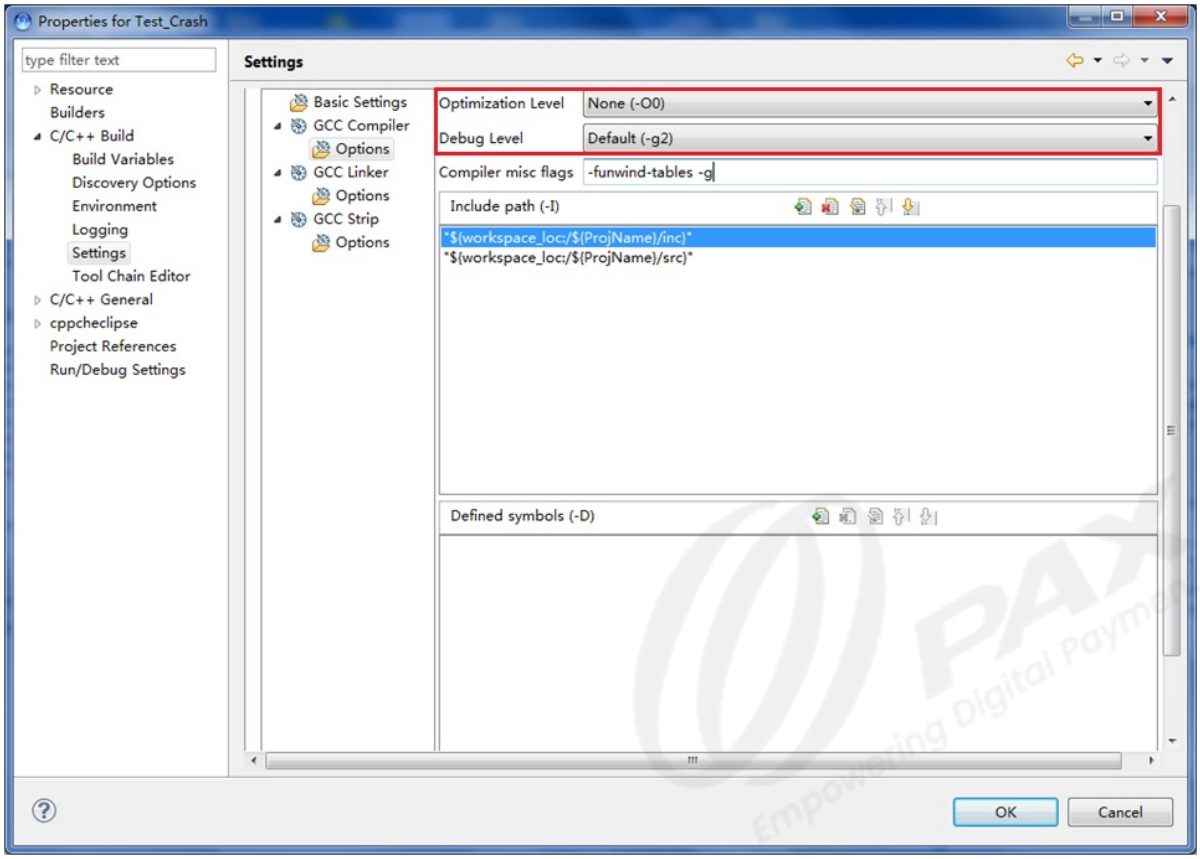
A crash report includes registers and stack information with symbol/function names when possible. Line numbers resolution is reserved for future use. To get symbols in unwound stack, build with -funwind-tables on ARM; and strip with -g switch if stripping binaries.The following is an example of crash report:

```
2012-06-23 11:17:34 | pid: 12345, tid: 12345  >>>
/path/to/crashed/program <<<
signal 11 (SIGSEGV), code 1 (SEGV_MAPERR), fault addr 00000000
 r0 0000000c  r1 00000000  r2 00000000  r3 0000000c
 r4 40165940  r5 00000000  r6 00000000  r7 00000000
 r8 00000000  r9 00000000  10 40024000  fp bebb0ad4
 ip bebb0ad8  sp bebb0ac0  lr 000087f0  pc 000087fc  cpsr 20000010
       #00  pc 000087fc  /path/to/crashed/program (parse_input)
```

```
        #01  pc 00008xxx  /path/to/crashed/program (listen_socket)
        #02  pc 00008xxx  /path/to/crashed/program (app_init)
        #02  pc 00008xxx  /path/to/crashed/program (main)
        #03  pc 00014fc8  /lib/libc-2.9.so (__libc_start_main)
    code around pc:
    000087dc ca000004 e51b3010 e2833001 e1a00003
    000087ec ebfffff3 ea000002 e3a02000 e51b3010
    ...
```

25.4 Notes

When programming with SDK, please be noted that “Optimization Level” should be configured as “None (-O0)” and “Debug Level” as “Default (-g2)”. The interface is as below.



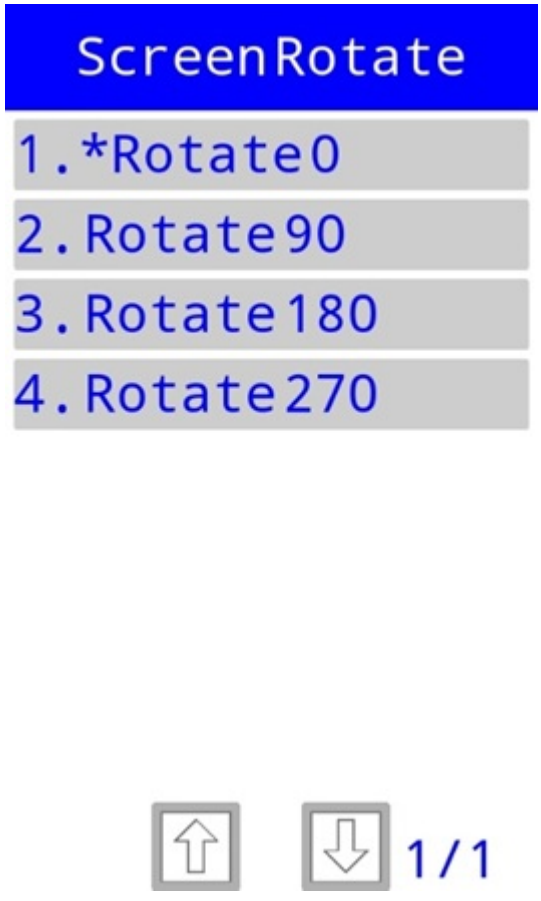
26 Screen Rotation

Currently, only the D220, Q20, IM20, and IM15 support screen rotation. For the TM interface, PED interface, and application interface, the D220 and IM15 support 90°, 180°, and 270° clockwise rotation, while the Q20 and IM20 only support 180° clockwise rotation. However, for the LOGO interface, rotation is not supported.

There are two ways to rotate the screen described as follows:

- **Set through TM**

  Go to TM->System Config->System Setting->Screen Rotate menu, the interface is shown as follows, select the rotation degree, and *Rotate 0* means no rotation. The setting will take effect after rebooting the terminal.



- **Set through Application**
  In the main application, you can set the value of the keyword "persist.sys.ui.rotate" to 0°, 90°, 180° or 270° to rotate the screen. For example, call `OsRegSetValue("persist.sys.ui.rotate", 90)` to rotate screen by 90° in clockwise direction. The setting will take effect after rebooting the terminal.

> NOTE
> When application needs to call `Xuiopen()`, set the parameter "ROTATE=" to the sum of keywords "ro.fac.lcd.rotate" and "persist.sys.ui.rotate".

27 LED Indicator

There are four types of LED indicator lights, which are magnetic stripe card reader (MSR) light, integrated circuit card (ICC) light, radiofrequency card (RF) light and RGB LED indicator lights. For PX7 model, there is also a LED light for the logo indicating where to tap the RF card. The operations of switching on/off all the indicator lights are implemented through their corresponding device nodes.

27.1 Device Node

1. **MSR indicator light:** Located at the swiping card area on the right side of a PX7, PX5 or Q30 model. There is one green light and one red light, and the corresponding device nodes are msr_green and msr_red respectively.
2. **IC indicator light:** Located at the bottom of password keyboard area of PX7, PX5 or Q30 model where you can swipe their IC cards. There is one red light and one green light, and the corresponding device nodes are icc_red and icc_green respectively.

3. **RF indicator light:** For PX7, Q80s, Q90, Q90s, Q30, Q50, QR55 and SP200 models, there are one blue light, one yellow light, one green light and one red light. The corresponding device nodes are rf_blue, rf_yellow, rf_green and rf_red respectively. For PX7 model, there is also a LED light for the logo indicating where to tap the card, and the corresponding device node is rf_logo.
4. **The RGB LED indicator lights:** Located at the card reader on Q20. There is one blue light, one green light and one red light, and the corresponding device nodes are rf_blue, rf_green and rf_red respectively. (Note: Although the name of device nodes contains “rf”, it is not specialized for “rf”.)

27.2 Operating Method

All device nodes of LED lights are in the “/sys/devices/platform/misc/” directory. To switch on/off a specified LED light, you need to write to the “power” node under the subdirectory of the corresponding LED light, to read the on/off state of LED light, you can read the “state” node of the corresponding LED light. For example: to turn on the red light of RF card, write “1” to “/sys/devices/platform/misc/rf_red/power”. On the contrary, to turn off the light, write “0” to the node.

The following is a sample code:

```
int  fd;
char state[32];
fd = open(“/sys/devices/platform/misc/rf_red/power”, O_WRONLY);
if (fd < 0)
{
   perror("open error");
return -1;
}
write(fd, “1”, 1);
close(fd);
```

```
fd = open("/sys/devices/platform/misc/rf_red/state", O_RDONLY);
if(fd < 0)
{
   perror("open error");
   return -1;
}
read(fd, state, sizeof(state));
printf("The state of rf_red is %s", state);
close(fd);
```

The following two tables show the paths of the "power" node and "state" node of LED light:

The path of the power node of LED light

LED	Related device	Node path	Value range and meaning
msr_red	PX5, PX7 and Q30	/sys/devices/platform/misc/msr_red/power	Write only, the value can be 0 or 1. • 0: off • 1: on
msr_green	PX5, PX7 and Q30	/sys/devices/platform/misc/msr_green/power	Ibid.
icc_red	PX5, PX7 and Q30	/sys/devices/platform/misc/icc_red/power	Ibid.
icc_green	PX5, PX7 and Q30	/sys/devices/platform/misc/icc_green/power	Ibid.

rf_blue	PX7, Q80s, Q90, Q90s, Q30, Q50, QR55 and SP200	/sys/devices/platform/misc/rf_blue/power	Ibid.
rf_yellow	PX7, Q80s, Q90, Q90s, Q30, Q50, QR55 and SP200	/sys/devices/platform/misc/rf_yellow/power	Ibid.
rf_green	PX7, Q80s, Q90, Q90s, Q30, Q50, QR55 and SP200	/sys/devices/platform/misc/rf_green/power	Ibid.
rf_red	PX7, Q80s, Q90, Q90s, Q30, Q50, QR55 and SP200	/sys/devices/platform/misc/rf_red/power	Ibid.
rf_logo	PX7	/sys/devices/platform/misc/rf_logo/power	Ibid.
red	Q20	/sys/devices/platform/misc/rf_red/power	Ibid.
green	Q20	/sys/devices/platform/misc/rf_green/power	Ibid.
blue	Q20	/sys/devices/platform/misc/rf_blue/power	Ibid.

NOTE

For magnetic stripe card and IC card, two different colors of the lights are in the same position. Therefore, when turning on one of the lights, the other light must be turned off in advance to ensure the terminal shows the correct color. For example, to turn on the red light of the magnetic stripe card, first write “0” to “msr_green” node, then write “1” to “msr_red” node, and vice versa.

- The path of state node of LED light

LED	Related device	Node path	Value range and meaning
msr_red	PX5, PX7 and Q30	/sys/devices/platform/misc/msr_red/state	Read-only, the value can be off or working. • off: LED light is off • working: LED light is on
msr_green	PX5, PX7 and Q30	/sys/devices/platform/misc/msr_green/state	Ibid.
icc_red	PX5, PX7 and Q30	/sys/devices/platform/misc/icc_red/state	Ibid.
icc_green	PX5, PX7 and Q30	/sys/devices/platform/misc/icc_green/state	Ibid.
rf_blue	PX7, Q80s, Q90, Q90s, Q30, Q50, QR55 and SP200	/sys/devices/platform/misc/rf_blue/state	Ibid.
rf_yellow	PX7, Q80s, Q90, Q90s, Q30, Q50, QR55 and SP200	/sys/devices/platform/misc/rf_yellow/state	Ibid.
rf_green	PX7, Q80s, Q90, Q90s, Q30, Q50, QR55 and SP200	/sys/devices/platform/misc/rf_green/state	Ibid.

rf_red	PX7, Q80s, Q90, Q90s, Q30, Q50, QR55 and SP200	/sys/devices/platform/misc/rf_red/state	Ibid.
rf_logo	PX7	/sys/devices/platform/misc/rf_logo/state	Ibid.
red	Q20	/sys/devices/platform/misc/rf_red/state	Ibid.
green	Q20	/sys/devices/platform/misc/rf_green/state	Ibid.
blue	Q20	/sys/devices/platform/misc/rf_blue/state	Ibid.

27.3 Color Combination

The three LED lights on the card reader of Q20 are in the same location, and they are controlled by independent IO interfaces. If a certain color of LED light is on, the corresponding color will be displayed; if two or more colors of LED lights are on at the same time, the corresponding combining color will be displayed. The combining color list is as follows:

LED light	Combining color
Red & Green	Yellow
Green & Blue	Cyan
Red & Blue	Magenta
Red & Green & Blue	White

28 Coulometer Battery Capacity

For POS terminal with coulometer module, when the battery is charging or discharging, applications can acquire the accurate battery capacity value through `OsCheckBMSMode()`. The battery capacity value ranges from 0 to 100. 0 indicates the battery has run out, if there is only battery supplying at this time, the terminal will shut down in 10s automatically; 100 indicates the battery is fully charged.

POS terminal with coulometer module includes D200 (V41 or higher version), S920 (V09 or higher version), D220, Q90, D190, D195, Q60, Q92, D199, D230, D270.

Application can call `OsRegGetValue()` to acquire keyword *ro.fac.coulomb_counter* value. If this value is non-zero, it means the terminal is integrated with the coulometer.

29 Application Auto-restart

If “rt.sys.mainapp.restart” is set to 1 by `OsRegSetValue()` in the application, the main application will restart automatically after exiting normally; if it is set to 0, POS will enter TM after the main application exits. The setting of “rt.sys.mainapp.restart” will be invalid after POS restarts, so if the application auto-restart function is needed, it must be set after POS restarts. Application auto-restart function is disabled by default.

30 XCB Service Auto-start

With “persist.sys.enable.debug” set to 1 by `OsRegSetValue()` in the application, if PED tampers or POS enters TM, XCB service will be started automatically; But in

PED tampers or POS enters TM, XCB service will be started automatically; But in the latter case, after entering the main application, XCB service will restore to the status it was before entering TM. With “persist.sys.enable.debug” set to 0, XCB service will not be started automatically under the two above-mentioned

31 Wireless Module Type

For terminals with a wireless communication module, the application can read “/var/posinfo/radio_type” file to acquire the type of wireless communication modules, such as GPRS, CDMA, WCDMA and LTE.

32 Logo Update

Prolin logo can be updated not only through the TermAssist tool and U disk download, but also by referring to the code encapsulated interface below. The sample code is as follows:

```
#include <unistd.h>
#include <errno.h>
#include "osal.h"
#ifndef ESIGNFAIL
#define ESIGNFAIL   130    /* Signature verification error
*/
#endif
intOsLogoUpgrade(const char *FileName)
{
const char *argv[16];
pid_tpid;
int status;
    if (FileName == NULL) {
        return ERR_INVALID_PARAM;
    }
    if (access(FileName, 0) < 0) {
```

```
        return ERR_FILE_NOT_FOUND;
    }
argv[0] = "installer";
argv[1] = "logo";
argv[2] = FileName;
argv[3] = NULL;
pid = fork();
    if (pid< 0) {
        return ERR_SYS_BAD;
    }
    else if (pid == 0) {
execvp("installer", argv);
        _exit(127);
    }
    else {
        while (waitpid(pid, &amp;status, 0) < 0) {
            if (errno != EINTR) {
                break;
            }
        }
    }
    status >>= 8;
    switch (status) {
    case 0:
        /* Success */
        return 0;
    case ENOENT:
        /*  The file doesn't exist */
        return ERR_FILE_NOT_FOUND;
    case EMEDIUMTYPE:
        /* File format error */
        return ERR_FILE_FORMAT;
    case EACCES:
        /* No permission */
        return ERR_ACCESS_DENY;
    case ESIGNFAIL:
        /* Signature verification error */
        return ERR_VERIFY_SIGN_FAIL;
```

```
    default:
        /* Error parameter */
        return ERR_INVALID_PARAM;
    }
    return ERR_SYS_BAD;
}
```

NOTE

For macros not defined in the above code, refer to the “Prolin API programming guide” or custom return values.

33 7-Zip Compression

The Prolin system includes the libarchive library, which supports archiving and compression in a variety of formats, including 7-zip compression. For more information, refers to http://www.libarchive.org/.

It is recommended that applications explicitly invoke the libarchive library and compile with the GCC link option with -ldl. In addition, 7-zip compression requires large memory space, and it may fail in D200, S900 and other terminals with low memory allocation. The sample code is as follows:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <fcntl.h>
#include <dlfcn.h>
#include "archive.h"
#include "archive_entry.h"
static void* archive_dp = NULL;
/* ------------------------libarchive API------------------------
*/
struct archive* (*archive_write_new_func)(void);
```

```
int (*archive_write_set_format_7zip_func)(struct archive *);
int (*archive_write_add_filter_none_func)(struct archive *);
int (*archive_write_set_format_option_func)(struct archive *,
    const char *, const char *, const char *);
int (*archive_write_open_filename_func)(struct archive *, const char *);
int (*archive_write_header_func)(struct archive *, struct archive_entry *);
__LA_SSIZE_T (*archive_write_data_func)(struct archive *, const void *,
size_t);
int (*archive_write_close_func)(struct archive *);
int (*archive_write_free_func)(struct archive *);
int (*archive_errno_func)(struct archive *);
const char* (*archive_error_string_func)(struct archive *);
struct archive_entry* (*archive_entry_new_func)(void);
void (*archive_entry_set_mtime_func)(struct archive_entry *, time_t, long);
void (*archive_entry_copy_pathname_func)(struct archive_entry *, const char
*);
void (*archive_entry_set_mode_func)(struct archive_entry *, __LA_MODE_T);
void (*archive_entry_set_size_func)(struct archive_entry *, __LA_INT64_T);
void (*archive_entry_free_func)(struct archive_entry *);
/* ------------------------------------------------------------------
-   */
/**
 * @param[in] outname: compressed file.
 * @param[in] filename: file that need to be compressed.
 * @param[in] compression_type:
 *   "bzip2", "copy", "deflate", "lzma1", "lzma2", "ppmd" or NULL.
 */
void write_7zip(const char *outname, char **filename,
    const char *compression_type)
{
    struct archive *a;
    struct archive_entry *entry;
    struct stat st;
    char buff[512] = {0};
    int len = 0;
    int fd = 0;

    a = archive_write_new_func();
```

```
    archive_write_set_format_7zip_func(a);
    archive_write_add_filter_none_func(a);
    if (compression_type != NULL &amp;amp;&amp;amp;
        ARCHIVE_OK != archive_write_set_format_option_func(a, "7zip",
        "compression", compression_type)) {
        printf("errno: %d, errstr: %s\n", archive_errno_func(a),
            archive_error_string_func(a));
    }
    archive_write_open_filename_func(a, outname);
    while (*filename) {
        stat(*filename, &amp;amp;st);
        entry = archive_entry_new_func();
        archive_entry_set_mtime_func(entry, st.st_mtime, 0);
        archive_entry_copy_pathname_func(entry, *filename);
        archive_entry_set_mode_func(entry, st.st_mode);
        archive_entry_set_size_func(entry, st.st_size);
        if (ARCHIVE_OK != archive_write_header_func(a, entry)) {
            printf("archive_write_header_func fail, errno: %d, errstr: %s\n",
                archive_errno_func(a), archive_error_string_func(a));
        }
        fd = open(*filename, O_RDONLY);
        memset(buff, 0, sizeof(buff));
        len = read(fd, buff, sizeof(buff));
        while (len > 0) {
            archive_write_data_func(a, buff, len);
            memset(buff, 0, sizeof(buff));
            len = read(fd, buff, sizeof(buff));
        }
        close(fd);
        archive_entry_free_func(entry);
        filename++;
    }
    if (ARCHIVE_OK != archive_write_close_func(a)) {
        printf("archive_write_close fail, errno:%d, errstr:%s\n",
            archive_errno_func(a), archive_error_string_func(a));
    }
    archive_write_free_func(a);
}
```

```
int load_libarchive()
{
    char *err;
    archive_dp = dlopen("/lib/libarchive.so", RTLD_LAZY);
    if(!archive_dp) {
        err = dlerror();
        printf("Can not load libarchive.so, err=%s\n", err);
        return -1;
    }
    archive_write_new_func = (struct archive* (*)(void))
        dlsym(archive_dp, "archive_write_new");
    archive_write_set_format_7zip_func = (int (*)(struct archive *))dlsym
        (archive_dp, "archive_write_set_format_7zip");
    archive_write_add_filter_none_func = (int (*)(struct archive *))dlsym
        (archive_dp, "archive_write_add_filter_none");
    archive_write_set_format_option_func = (int (*)(struct archive *,
        const char *, const char *, const char *))
        dlsym(archive_dp, "archive_write_add_filter_none");
    archive_write_open_filename_func = (int (*)(struct archive *, const char *))
        dlsym(archive_dp, "archive_write_open_filename");
    archive_write_header_func = (int (*)(struct archive *, struct archive_entry
*))
        dlsym(archive_dp, "archive_write_header");
    archive_write_data_func = (__LA_SSIZE_T (*)(struct archive *, const void *,
        size_t))dlsym(archive_dp, "archive_write_data");
    archive_write_close_func = (int (*)(struct archive *))
        dlsym(archive_dp, "archive_write_close");
    archive_write_free_func = (int (*)(struct archive *))
        dlsym(archive_dp, "archive_write_free");
    archive_errno_func = (int (*)(struct archive *))dlsym(archive_dp,
"archive_errno");
    archive_error_string_func = (const char* (*)(struct archive *))
        dlsym(archive_dp, "archive_error_string");
    archive_entry_new_func = (struct archive_entry* (*)(void))
        dlsym(archive_dp, "archive_entry_new");
    archive_entry_set_mtime_func = (void (*)(struct archive_entry *, time_t,
        long))dlsym(archive_dp, "archive_entry_set_mtime");
    archive_entry_copy_pathname_func = (void (*)(struct archive_entry *,
```

```
        const char *))dlsym(archive_dp, "archive_entry_copy_pathname");
    archive_entry_set_mode_func = (void (*)(struct archive_entry *,
__LA_MODE_T))
        dlsym(archive_dp, "archive_entry_set_mode");
    archive_entry_set_size_func = (void (*)(struct archive_entry *,
__LA_INT64_T))
        dlsym(archive_dp, "archive_entry_set_size");
    archive_entry_free_func = (void (*)(struct archive_entry *))
        dlsym(archive_dp, "archive_entry_free");
    if (!archive_write_new_func || !archive_write_set_format_7zip_func ||
        !archive_write_add_filter_none_func ||
        !archive_write_set_format_option_func ||
        !archive_write_open_filename_func || !archive_write_header_func ||
        !archive_write_data_func || !archive_write_close_func ||
        !archive_write_free_func || !archive_errno_func ||
        !archive_error_string_func || !archive_entry_new_func ||
        !archive_entry_set_mtime_func || !archive_entry_copy_pathname_func ||
        !archive_entry_set_mode_func || !archive_entry_set_size_func ||
        !archive_entry_free_func) {
        printf("dlsym error\n");
        return -1;
    }
    return 0;
}
int main(int argc, char *argv[])
{
    const char *outname;
    const char *compress_type;

    if (argc < 4) {
        printf("[usage] %s [compression type] [outname] [pathname]\n",
argv[0]);
        printf("[compression type]: bzip2, copy, deflate, lzma1, lzma2,
ppmd\n");
        return 0;
    }

    argv++;
```

```
ppmd\n");
        return 0;
    }

    argv++;
    compress_type = *argv++;
    outname = *argv++;
    if (load_libarchive() == 0) {
        write_7zip(outname, argv, compress_type);
```

34 US_PUK Information

Open the /sys/pax/puk_list file in read-only mode to read the installed US_PUK file on the system. One line of information is divided into three fields, each field is separated by a semicolon ":", respectively representing the slot of PUK, CRC checksum value and owner which can be empty. For example, after downloading the US_PUK of PAX to slot 0, the PUK information is "0:0xA40EB99B:PAX" (ending with line break 0x0A). When you download multiple US_PUK, you have multiple lines of information, as shown below:

0:0x4A123456:
1:0xD9123456:PAX
2:0x63123456:PAX

35 USB OTG Device Detection

When the POS terminal needs to convert USB OTG to RS232 to communicate with other devices, the device node is not fixed because its transmission protocol is the same as the USB transmission protocol of the 4G or 3G module. Therefore, detection is needed when selecting the device node. The detected device name is used as the device node. Taking S920 as an example, the following provides an example code for detecting a device node:

```
#define PATH "/sys/devices/platform/dwc_otg/usb1/1-1/1-1:1.0"
int GetSerialName(char* SerialName)
{
    int i;
    char original_name[128] = PATH;
    char findout_flag = 0;
     char temp_str[24] = {0};
    char temp_path[156] = {0};
    for (i = 0; i <= 5; ++i) {
        memcpy(temp_path, original_name, strlen(original_name));
        snprintf(temp_str, sizeof(temp_str)-1, "/ttyUSB%d", i);
        strcat(temp_path, temp_str);
        if (access(temp_path, F_OK) == 0) {
            findout_flag = 1;
            break;
        }
        memset(temp_path, 0, sizeof(temp_path));
        memset(temp_str, 0, sizeof(temp_str));
    }
    if (findout_flag) {
        memset(original_name, 0, sizeof(original_name));
        memcpy(original_name, temp_path, strlen(temp_path));
    }
    else {
        return -1;
    }
    snprintf(SerialName, sizeof(SerialName)-1, "/dev%s", temp_str);
    return 0;
}
```