

API User Manual

Table of Contents

1. BASIC DATA STRUCTURE _____	3
2. FORM DATA STRUCTURE _____	4
3. FORM CONSTANTS _____	17
4. FORM RELATED FUNCTIONS _____	20
5. GUI INTERNAL FUNCTIONS _____	47
6. MESSAGE QUEUE DEFINITIONS AND RELATED FUNCTION__	50
7. DATABASE API _____	57
8. OTHER FUNCTIONS _____	67
9. STANDARD LINUX LIB FUNCTIONS _____	69

API User Manual

Since we develop the GUI ourselves. We have to define every function, data structure even the message queue. But, our target is to 'binary compatible' Palm Application, maybe we'll improve our API to follow Palm SDK later.

In this version of API, We call each form as 'FORM', that means the main windows of a application is a form, a message box is also a form. (I know Palm SDK treat every window as 'Form'). Currently, our applications are 'FORM based' app. And, in each form, we call the elements such as 'Button', 'Label' as 'Components'.

FORM

This section provides the following information about FORM

1. Basic data structure
2. FORM data structure
3. FORM constants
4. FORM related functions
5. GUI Internal functions
6. Message queue definitions and related functions
7. A simple example to show how to use the GUI functions to get the message

1. Basic Data Structure

These data structure are defined in 'gui.h'

```
typedef unsigned char U8;  
typedef unsigned short U16;  
typedef unsigned long U32;
```

```
typedef struct {           //define a pixel, (x,y) is co-ordinates of the pixel  
    short x;  
    short y;  
}PIXEL;
```

```
typedef struct {  
    short left;           //the x coordinate of the start point;
```

```
short top;          //the y coordinate of the start point;
short right;        //the x coordinate of the end point;
short bottom;      //the y coordinate of the end point;
}RECT;
```

2. FORM Data Structure

These data structure are defined in 'gui.h'

```
typedef struct {
    U8 type;          //component types
    U8 id;            //the component id , when some component is clicked ,the id will be returned
    U16 x,y,w,h;     //(x,y) is the left-top corner co-ordinates of the component ,
                    //and "w" is width, "h" is height
    void *text;      //a pointer, may have different meaning
    U8 style;        //special styles
    U8 status;       //the component is "visible" or "invisible"
    U8 font;         //each component has its own font. Now there are three types of fonts: small,
                    //medium and large. Recommend to use small fonts because the screen size
                    //is rather small. The definitions of the font style are list below.
}FORM;
```

```
// font types definition
#define FONT_SYS      0

#define FONT_SMALL 0
#define FONT_MEDIUM 1
#define FONT_LARGE 2
```

Field descriptions

Type	Component type.
Id	Each component in a form has it's own id. When some component is clicked, the id will be returned .
X,y,w,d	The co-ordinates of each component, the value (x,y) is related to the left-top corner of it's own form.. They are offset coordinates to the form left-top corner..
*text	It's only a pointer, for different components, it has different meaning. And this value can be NULL. <i>CAPTION:</i> pointer to FORMCAPTION structure. <i>LABEL:</i> the string of the label <i>BITMAP:</i> the pointer of BITMAP structure

	<i>MENU:</i>	the display string of the menu
	<i>MENUITEM:</i>	the display string of menu item
	<i>CHECKBUTTON:</i>	the pointer of CHECKBUTTON structure
	<i>SCROLLBAR:</i>	the pointer to SCROLLBAR structure.
	<i>TEXTBOX:</i>	the pointer to TEXTBOX structure.
	<i>SPINBUTTON:</i>	the pointer to SPINBUTTON structure
Style		Indicates the special styles of each kind of component
Status		Indicates the component is 'visible' or 'invisible'
Font		Specify the Font of the string in the component .There is 3 types font in our GUI.They are defined in "mini_gui.h".

The below is the simple description of our currently components, you can find details and examples in the behind:

{FORM_CAPTION,0,0,0,159,159, &caption,0,COMPONENT_VISIBL E,FONT_SMALL },

This is the 1st item of your form definition. *The field 'type' must be FORM_CAPTION.*

Here, we want our form's rectangular region is (0,0) to (159,159), and the caption of the form is 'About us', which you can choose to show or not. Of course you can show a form not so big. In this case, the field 'id' can be set to any value. If we set 'style' to FORM_FRAME_BORDER, the form will shown as a no border form. If the bit 'FORM_SAVEBEHIND' in this field is set, system GUI will back up the background picture before the new FORM display ;and, the background of the FORM will be restore after function 'CloseForm' is invoked.

There are other styles you can refer to behind details.

{FORM_BUTTON,1,70,130,45,20," Accept",0,COMPONENT_VISIBLE ,FONT_SMALL },

This item defines a standard button. String "Accept" will be shown as the text of the button, the text in each button will be automatic align to the middle and center of the rectangle area of the button .

Field 'id': while click the button, function GetMessage() will return the id of the button.

Field 'status': COMPONENT_VISIBLE will set the button as 'visible'

{FORM_MENU,20,45,20,0,0,"ABC D",0,COMPONENT_VISIBLE,FO NT_SMALL }

This item indicates a menu whose id is '20' will be shown and the pixel (45,20) is it's left-top corner. for this type component, the width and the height will not available, COZ my GUI will calculate the width and height automatically. Here we set the width & height as 0.

The field 'text' means that the text string of menu is 'ABCD'.

{FORM_MENUITEM,21,5,20,0,0," Digitizer",0}

Each menu should has it's sub-menu. In most cases, the sub menu will be drop down while click in the menu.

For this type of component, the fields x,y,w,d are all not

available. All of these fields are calculated by GUI, although you can set the value of these fields, they will NOT take effect while these items are shown. COZ these fields are overlay by GUI at that time.

If u want to define some sub menu items of one menu, u MUST place these sub-menu item definitions after its parent menu!

```
{FORM_BITMAP,40,0,18,11,140,in  
k_tb,BITMAP_BUTTON,COMPON  
ENT_VISIBLE},
```

Maybe it's the most complex and most powerful component in our GUI.

First of all, we can use this item to display a uncompressed bitmap in our UI. Please notice the field 'text', it's NOT a true text string pointer now! it's the pointer of a structure BITMAP. The definition of BITMAP is:

```
Typedef struct {  
    short width,height;  
    void *img;  
    char *caption;  
}BITMAP;
```

in this structure, field width and height are the size of bitmap image, and the pointer *img points to the data of the bitmap, field 'caption' is not always useful, sometime when we need to display a bitmap with a string, the field becomes useful.

The other way, have you thought about an 'Owner draw' component? Maybe you hate the standard components, Do you feel them ugly and want to draw one yourself? If yes, you can use this component to reach your target. Because the GUI do noting while the pen click in, u can use the returned message to do everything you want, for example, the event responses function. Anyway, u can use this kind of component to create your own button, the check button , the radio button, the spin button, the scroll bar...even any component u can imagine.

While the pen click in this component ,a message "EVENT_CLICK" will be returned; if the pen is pulled up ,message "EVENT_PEN_UP" will be returned.

```
{FORM_END}
```

Here is the last line of the form definition.
Please remember: this line is necessary!

Following are the comprehensive descriptions of all members we now have in FORM:

FORM_CAPTION

This is the 1st item in your form definition. *The field 'type' must be FORM_CAPTION.* You can choose show it in different styles or not.

type ---- must be FORM_CAPTION;

id ---- can be set to any value, because it has no effect .

x,y,w,h -- (x,y) is the coordinate of the left-top point the form ,and w is its width,
h is its length.

*text ---- point to the struct FORMCAPTION which includes the caption content of the form.

```
typedef struct {
    char *str;    //the form caption is a string;
    void *icon;  //the form caption is an icon;
    RECR r;     //reserved, system used.
}FORMCAPTION;
```

style ---- related to how the FORM is shown. Now there are four styles as follows. A form can have one or more styles. You can use “|” operand to connect them.

1. FORM_FRAME_CAPTION

If this style is set, a caption Bar will be shown in the head of the form. As the Figure 1.2.1 shown, “LinuxDA” is the form caption.If a form only has the style, the caption include the black rectangle where the caption string display and the line under it. The blanket area on the left behind the rectangle can hold other components such as buttons, combobox . In Figure 1.2.1 the black bar on the center and the time string on the right do not belong to the form caption.



Figure 1.2.1



Figure 1.2.2



Figure 1.2.3

2. FORM_FRAME_BORDER

A form with black border will be shown. As Figure 1.2.3 the small form in the center

has border.

3. *FORM_DLGF*FRAME

When this style is given to a form, the form caption will show as Figure 1.2.2. The caption rectangle where “Message” show is as wide as the form width.

4. *FORM_SAVE*BEHIND

Save the background while a new FORM is created and its background will be restored if function “CloseForm()” is invoked.

Recommend not to set a form to this style unless it is necessary, for instance , to show such a form as the messagebox as Figure 1.2.2 shown..

Caution: DO NOT Forget to invoke “CloseForm” before creating another FORM if the form has style FORM_SAVEBEHIND, otherwise the memory that the saved background image occupied will not free and it may cause error.

There is another way to fulfill the same effect as SAVE_BEHIND. You can apply a block of memory and use the pair functions ‘CopyScreen()’ and ‘PasteScreen()’. Use ‘CopyScreen()’ to save the background into the memory, and let ‘PasteScreen()’ to recover the background and free the memory. This is a safer way than using style SAVE_BEHIND;

If you set the style to zero,the caption will not be shown.

status --- the status of a caption need not to consider, for it has no meaning for the FORM_CAPTION .

font ----- what font you like. FONT_MEDIUM is commonly used in the caption.

The FORM_CAPTION in Figure 1.2.1 is defined in a form definition as below:

```
FORMCAPTION form_caption={"LinuxDA"};
FORM frmMain[]={
{FORM_CAPTION,0,0,0,159,159,&form_caption,FORM_FRAME_CAPTION,COMPONENT_VISIBLE,
FONT_MEDIUM},
.....
{FORM_END}
};
```

The FORM_CAPTION in Figure 1.2.2 is defined in the form definition as below:

```
FORMCAPTION form_cap={"Message"};
{FORM_CAPTION,0,20,20,80,60,&form_cap,FORM_FRAME_CAPTION|FORM_FRAME_BORDER|
FORM_DLGF|FORM_SAVEBEHIND,COMPONENT_VISIBLE,FONT_MEDIUM},
```

FORM_LABEL



Figure 1.2.4 Labels

Labels are often used before other components to indicate what the meanings of them. In Figure 1.2.4, “Last Name” and “First Name” are two labels.

type ---- set to FORM_LABEL

id ----- set a value .

x,y,w,h -- they are all available

*text ---- point to the label string

style ---- can set how the LABELS is aligned and shown. There are 6 types of the alignment method to text which you can find behind. You must set both x and y alignments, or the label may not be properly shown.

Other styles:

LABEL_BORDER-----The label has a border and this will make the label look like a button.

LABEL_MULTILINE---The label has multiple lines. Each line is separated by ‘\n’.

status ---- set it invisible or visible.

font ----- which font you use to show the label.

The FORM_LABEL in Figure 1.2.4 is defined in the form definition as below:

```
{FORM_LABEL, 10, 5, 5, 30, 15, "Last Name", X_ALIGN_RIGHT|Y_ALIGN_CENTER,  
COMPONENT_VISIBLE, FONT_SMALL},
```

```
{FORM_LABEL, 20, 5, 25, 30, 15, "First Name", X_ALIGN_RIGHT|Y_ALIGN_CENTER,  
COMPONENT_VISIBLE, FONT_SMALL},
```

FORM_MENU

Shown in the top of the form. A menubar is commonly hidden unless you either click the region which the menu will next be shown in (This region may be superposed by the caption of the form), or press the hardware button which is designed to invoke the showing of the menu. “Tools” in the Figure 1.2.4 is a menu in the top menu bar.

The event EVENT_SHOW_MENUBAR will be returned when the menubar is shown . And The event EVENT_HIDE_MENUBAR will be cause when the menubar is hidden.

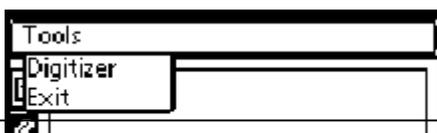


Figure 1.2.5

type ----- set to FORM_MENU
id ----- can be set a value as you want to
x,y,w,h -- (x,y) as usual, but the w ,h is invalid ,GUI will calculate the width and the height automatically, and any values given will take no effect.
*text ---- point to a text string which will be shown on the menu;
style----- commonly set to "0"
status --- set it to invisible or visible.
font ----- which font to show the menu's text. Usually we use FONT_MEDIUM.

You can refer to its usage in the followed example in the introduction of FORM_MENUITEM.

FORM_MENUITEM

The items are shown in the rectangle the menu is click .the "Digitizer" and "Exit" are two menu items in Figure 1.2.5.

type ----- set to FORM_MENUITEM
id ----- can be set a value as you want to, but remember not same as the other components' id.
x,y,w,h -- x, y, w, h is not available. All of these are calculated by GUI.
*text --- point to a string which will be shown on the menuitem when the menu is pulled down.
style----- commonly set to "0" for there is no style for it.
status ---- usually COMPONENT_VISIBLE.
font ----- with which font to show the menuitem's text. Usually we use FONT_MEDIUM.

Caution: if you want the menu items show in a menu, you must let these menuitems follow the menu immediately.

The FORM_MENU and FORM_MENUITEM in Figure 1.2.5 are defined in the form definition as below:

```
FORM frm[]={  
.....  
{FORM_MENU,10, 5, 20, 0, 0, "Tools",0, COMPONENT_VISIBLE,FONT_SMALL},  
{FORM_MENUITEM, 11,0, 0, 0, 0, "Digitizer",0, COMPONENT_VISIBLE,FONT_SMALL}  
{FORM_MENUITEM, 12,0, 0, 0, 0, "Exit",0, COMPONENT_VISIBLE,FONT_SMALL}  
.....  
};
```

FORM_BUTTON

It is a very common component and you may often use. Now there are only the standard rectangle buttons that the face of it is a string.

type ----- set to FORM_BUTTON.

id,x,y,w,h ---- all are available.

*text ----- point to a string which will be shown on the button.

style ----- you can set BUTTON_NO_BORDER if you hope to show a button has no border.

status ----- "0" to invisible, and "COMPONENT_VISIBLE" for visible.

font ----- which font to show the text of the button.

The FORM_BUTTON in Figure 1.2.2 is defined in the form definition as below:

```
{FORM_BUTTON,40,20,20,40,20,"Ok!",0,COMPONENT_VISIBLE,FONT_SMALL},
```

FORM_BITMAP

type ----- FORM_BITMAP

id, x, y, w, h -- they are all available ,and w is the bitmap's width, and the h is its height.

*text ----- point to a structure BITMAP, you can find details in the definition.

Style ----- you can set BITMAP_BORDER for showing a border outside the bitmap. You can set BITMAP_BUTTON and then when clicked ,the event will delay to happen when the pen up, that means the bitmap will act as a button.

status ----- the same as button.

font ----- which font you use to show the text of the button

In the Figure 1.2.6, each button is a bitmap, for example the "9" button is realized as follows:

```
#define b9_width 29
```

```
#define b9_height 28
```

```
unsigned char b9_tb_bitmap[]={
```

```
0x00,0x1F,0xC0,0x00,0x00,0xFF,0xF8,0x00,0x03,0xFF,0xFE,0x00,0x07,0xC7,0xFF,  
0x00,0x0F,0x1F,0xFF,0x80,0x1E,0x7F,0xFF,0xC0,0x3C,0xFF,0xFF,0xE0,0x39,0xF0,  
0x7F,0xE0,0x73,0xC0,0x1F,0xF0,0x77,0xC2,0x1F,0xF0,0x67,0x87,0x0F,0xF0,0xEF,  
0x87,0x0F,0xF8,0xEF,0x87,0x0F,0xF8,0xFF,0xC1,0x0F,0xF8,0xFF,0xE0,0x0F,0xF8,  
0xFF,0xF8,0x1F,0xF8,0xFF,0xFE,0x1F,0xF8,0x7F,0xFC,0x3F,0xF0,0x7F,0xF8,0x3F,  
0xF0,0x7F,0xF0,0x7F,0xF0,0x3F,0xF9,0xFF,0xE0,0x3F,0xFF,0xFF,0xE0,0x1F,0xFF,  
0xFF,0xC0,0x0F,0xFF,0xFF,0x80,0x07,0xFF,0xFF,0x00,0x03,0xFF,0xFE,0x00,0x00,  
0xFF,0xF8,0x00,0x00,0x1F,0xC0,0x00};
```

```
// The array is got by a tool which can convert a tiff picture file to a text file.
```

```
const BITMAP b9[]={b9_width, b9_height, b9_tb_bitmap};
```

//In the Figure 1.2.6 form the FORM_BITMAP “9” is defined as follows

```
static FORM frmBasic[]={
.....
{FORM_BITMAP,100,1,40,b9_width,b9_height,b9,0,COMPONENT_VISIBLE,},
.....
};
```

FORM_CHECKBUTTON

The checkbutton is such a component as the “ I am CheckButton” shown in Figure 1. 2. 6. The check mark in the square indicates the check button is chosen.

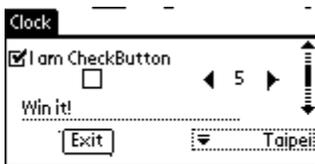


Figure 1.2.6

The FORM_CHECKBUTTON is more complicate than the above components. This component is composed of a and a string.

```
type ----- FORM_CHECKBUTTON;
id ----- set a value as you want;
x, y ----- the coordinate of left-top point.
w, h ----- let them be. System will generate the correct values according to the
             height and width of the string;
*text --- point to a CHECKBOX structure, which is defined in “gui.h”
typedef struct{
    short value; // it will be “0” for not chosen status and “1” for chosen status.
                //if you want to get the checkbutton’s status you only need to know //what
                the value is.
    void *caption; // the caption string will be shown following the checked button.
}CHECKBUTTON;
style ----- set how the checkbutton is shown;
status --- “0” to invisible, and “COMPONENT_VISIBLE” for visible;
font ----- which font you use to show the caption;
```

The FORM_CHECKBUTTON in Figure 1.2.6 is defined in the form definition as below:

```
CHECKBUTTON checkbutton = {0,“I am CheckButton”};
{FORM_CHECKBUTTON,10,2,2,40,15,&checkbutton,0,COMPONENT_VISIBLE,FONT_SMALL
```

}

FORM_SPINBUTTON

Now we have the vertical and horizontal style SPINBUTTONs. If it is horizontal style, a spinbutton can hold a string between two arrows. As Figure 1.2.6 show, “5” is in the middle of the left and right arrow of the horizontal spinbutton. But vertical spinbutton can not. This component does not store any value like the position of SCROLLBAR. A value may be “-1” or “+1” returned in the element “value” of data type GUI_MSG when it is clicked.

```
type ----- FORM_SPINBUTTON;
id,x,y ---- set as you need;
w,h ----- let these two members be, system will set the w 11,d to 14;
*text --- point to the below structure;
        typedef struct {
                short spacing;      //you can leave some space between the two arrows.
                char str[32];       // if HSPINBUTTON, you can show text between
                                   // the two arrows.
        }SPINBUTTON;
style ----- VSPINBUTTON or HSPINBUTTON;
statue --- visible or invisible;
font ----- which font you use to show the text of the button;
```

Note : while clicking the member “value” of message returned will be “-1” or “+1”.

The FORM_SPINBUTTON in Figure 1.2.6 is defined in the form definition as below:

```
SPINBUTTON spinbutton={ 15, “5”};
{FORM_SPINBUTTON,10,90,30,20,15,&spinbutton,HSPINBUTTON,COMPONENT_VISIBLE,FO
NT_SMALL}
```

FORM_SCROLLBAR

There are two types of the SCOLLBAR, one is vertical scrollbar, the other is Horizontal scrollbar. As the image in the right of Figure 1.2.6 shown, a scroll bar includes two arrows and one page. It can store the position of this page.

The Related macros in “gui.h”

```
#define VSCROLLBAR 0
#define HSCROLLBAR 1
type ----- SCROLLBAR;
id,x,y --- give values as you want to;
w ----- for Vertical SCROLLBAR this member is NOTUSED, because system will be set
it to 7;
h ----- for Horizontal SCROLLBAR this member is NOTUSED, because system will be
```

```
set it to 7
*text --- the pointer of a structure SCROLLBAR which is defined in "gui.h"
typedef struct{
    short    max, position, page_height;
    /* max : The maximum times the bar can scroll, and it is not the
           max of pixel number. It set the limitation that the position can
           changed.
           Position: The current position of the bar. It can changed in the
           range form 0 to max-1.
           Page_height: It means the amount that the variable position will
           be added or subtracted when the page scrolls down or up.
           If you set this value greater than 'max', the system will adjust
           it to a suitable value. If you really don't understand what the
           meaning is of this member, set it to 0 and then the system will
           give a default value.
    */
    RECT    first_r, second_r, third_r, last_r, button_r; /*system used. Application
           programmer need not care about them. */
    style---- should be HSCROLLBAR or VSCROLLBAR;
    status ---- simply set it to visible or invisible;
    font ---- Not valid;
```

The FORM_SCROLLBAR in Figure 1.2.6 is defined in the form definition as below:

```
SCROLLBAR scollbar={ 10,0,20,};
{FORM_SCROLLBAR,50,140,5,5,40,&scollbar,BSCOLLBAR,COMPONENT_VISIBLE,FONT_S
MALL}
```

FORM_COMBOBOX

A combobox is somewhat like the component menu which also has a couple of items. It displays currently selected item. The list box is popped when the combobox is clicked and invoked. If an item is selected from the list, the popup will be closed and the selected item will be reflected in the combobox. When the items are larger than the domain of list, a spinbutton can be added automatically for scrolling the items.

As the Figure 1.2.6 shown, when click the "Basic" combobox, a list including some string items will be shown.

```
type ---- FORM_COMBOBOX
id ----- set a value.
x, y, w, h-they are all available
```



Figure 1.2.6

*text --- point to the below COMBOBOX structure:

```
typedef struct {
    short index;    //maybe you can sort the items when shown by this field.
    char str[32];  //the text of the item.
}ITEM;
typedef struct {
    short max_items; /* the max items the list can have.*/
    short selected;  /* which item is current selected. It counts form 0 and it is
                       decided by the location of the item and not the item index*/.
    short drop_down_width, drop_down_height;
                       /*the listbox size when it drops down*/.
    ITEM items;      /* refer to the above structure. */
}COMBOBOX;
```

style ----- Four style you can choose to show the component:

COMOBOBOX_NO_BORDER: Remove the border mad up of broken lines of the
ComboBox;

COMBOBOX_LEFT_ALIGN: Let the string in the Combobox be shown next to its left
border and the arrow also in the left;

COMBOBOX_RIGHT_ALIGN: Let the string in the Combobox be shown next to its
right border and the arrow also in the right;

COMBOBOX_NO_ARROW: Remove the arrow.

status --- set it invisible or visible.

font ----- which font you use to show the items of the ComboBox.

The FORM_COMBOBOX in Figure 1.2.6 is defined in the form definition as below:

```
ITEM items[]={0,"Basic"},{1,"Scientific"},{2,"Financial"},{3,"Exit"};
COMBOBOX combobox={10,0,40,60,items};
{FORM_COMBOBOX, 60, 120, 3, 40, 60, &combobox, 0, COMPONENT_VISIBLE,
FONT_SMALL};
```

FORM_TEXTBOX

Textbox is a component can hold the content user inputs. Since its content and then its size frequently changed, its refreshing procedure shows some intricacy. When textboxes are used in a form combined with other components, an inattention will make the form disarray. A textbox can be line-changed, and you need to carefully read the textbox details and try it for some times to be familiar with textbox.

```
type ---- FORM_TEXTBOX
id ----- set a value.
x, y ---- the left-top point of the textbox
w ----- the textbox width. Remember each line of a multiple lines textbox has the same
           width for it is aligned with the previous line.
h ----- The whole height of the textbox;
           Any value you set is invalid for it is controlled by system. You can get it when you
           need this value. Remember each line may has different height for the font height of
           different letters are not the same.
           When you change the content of the textbox, the height of the textbox will be changed
           correspondingly. You can get the changed height value of the textbox from GUI_MSG
           structure when you ended inputting (EVENT_INPUTCHAR). The value in the structure
           of this GUI_MSG whose event is EVENT_INPUTCHAR is the offset value of the 'h' in
           pixel unit.
*text --- point to the below structure
           typedef struct {
               short counter_lines;    //reserved
               char string[256];       // the string shown in the textbox;
               short max_char;         //the maximum character amount the textbox can
                                       hold.
               short min_empty_lines; //the minimum lines a textbox can be displayed in the
                                       screen when it is invisible.
               short start_line;       //the line in the textbox will shown first in the screen.
               short visible_lines;    //how many lines of the textbox will be visible in the
                                       screen.
           }TEXTBOX;
style ---- let it be.
status --- set it invisible or visible.
font ---- which font you use to show the label. (FONT_SMALL, FONT_MEDIUM,
           FONT_LARGE)
```

The FORM_TEXTBOX in Figure 1.2.6 is defined in the form definition as below:

```
TEXTBOX textFirstName={3,"Toris",32,1,0,-1};
```

```
TEXTBOX textLastName={3,"Chang",32,1,0,0};
{FORM_TEXTBOX,10,35,5,60,0,&textFirstName,0,COMPONENT_VISIBLE,FONT_SMALL}
{FORM_TEXTBOX,20,35,25,60,0,&textLastName,0,COMPONENT_VISIBLE,FONT_SMALL}
```

FORM_END

It is the last line of the form definition. Pls remember: this line is necessary!

```
type-----be FORM_END
id,x,y,w,h,*text,style,status,font-----let them be.
```

3. FORM Constants

These Marcos are defined in 'gui.c'

```
/******
 *
 *   Basic FORM elements type definitions
 *
 *****/
#define FORM_CAPTION      1
#define FORM_LABEL        2
#define FORM_BUTTON       3
#define FORM_BITMAP       4
#define FORM_MENU         5
#define FORM_MENUITEM     6
#define FORM_SYSTEM       7
#define FORM_SCROLLBAR    8
#define FORM_TEXTBOX      9
#define FORM_SPINBUTTON   10
#define FORM_CHECKBUTTON  11
#define FORM_COMOBOX      12
#define FORM_END          127

typedef struct {
    short width,height;    //the size of the bitmap image
    void *img;             //point to the data of the bitmap, usually the first address of array
                           //which expresses a bitmap
    char *caption;
```

```
}BITMAP;
/*****
*
*   For each component,bit 7
*   is a 'Visible' flag
*
*****/
#define COMPONENT_VISIBLE 0x80
#define COMPONENT_INVISIBLE ~0x80
/*****
*
*   Button component style
*
*****/
#define BUTTON_NO_BORDER 0x01    //remove the border while display the button
/*****
*
*   Bitmap component style
*
*****/
#define BITMAP_BORDER      0x02    //show a border while display the bitmap
#define BITMAP_BUTTON      0x04    //use the bitmap as button. It will process events
                                   //when the pen up
/*****
*
*   FORM Frame Styles
*
*****/
#define FORM_FRAME_CAPTION  0x01
#define FORM_FRAME_BORDER  0x02
#define FORM_SAVEBEHIND    0x04    //Auto save the background
#define FORM_DLGFRAME      0x08    //Caption width as same as the form.
/*****
*
*   Text alignment methods
*
*****/
#define X_ALIGN_LEFT        0x01
#define X_ALIGN_MIDDLE      0x02
#define X_ALIGN_RIGHT       0x04
#define Y_ALIGN_TOP         0x08
```

```
#define Y_ALIGN_CENTER    0x10
#define Y_ALIGN_BOTTOM   0x20
/*****
*
*   Label styles
*
*****/
#define LABEL_BORDER      0x40
#define LABEL_MULTILINE  0x80
/*****
*
*   The MENU status
*
*****/
#define MENU_DROPPED      1
#define MENU_UNDROPPED   0
/*****
*
*   SCROLLBAR styles
*
*****/
#define VSCROLLBAR        0
#define HSCROLLBAR        1
/*****
*
*   SCROLLBAR styles
*
*****/
#define VSPINBUTTON       0
#define HSPINBUTTON       1
/*****
*
*   COMBOBOX styles
*
*****/
#define COMBOBOX_NO_BORDER    1
#define COMBOBOX_LEFT_ALIGN  2
#define COMBOBOX_RIGHT_ALIGN 4
#define COMBOBOX_NO_ARROW    8
```

4. FORM Related Functions

in *“mini_gui.h”*

void init_screen(void);

Initialize the related devices--- the drivers of LCD & Digitizer

Sample Code:

```
/*
 *This program is to demo how to use GetLaunchCommand(), PutAppInfo( ) init_screen(),
close_screen(). This is the main program.
 * GetLaunchCommand() to parse the input parameter in command line. Using its return value to
decide to run the normal program or run PutAppInfo( )
 *Normal program use init_screen() to initial the form , Exit the form use close_screen()
 */
void main(int argc,char * argv[])
{
    short launch_cmd;

    launch_cmd=GetLaunchCommand(argc,argv);
    switch(launch_cmd)
    {
        case CMD_RUN_NORMAL :
            init_screen();
            Sample();
            close_screen();
            break;
        case CMD_RUN_GET_INFO:
            PutAppInfo(&sample_info);
            break;
    }
};
```

void close_screen(void);

Release the screen related devices.

These above two functions is so low_level that applications needs not to invoke them.

Sample Code:see void init_screen(void)

void ClearMessageQueue(void);

Clear the message queue. It is usual to clear the message queue when a new form is created. You must get messages from message queue before processing events.

Sample Code:

```
/*
```

*This program demos the following API. ClearMessageQueue(), GetMessage() hibyte() lobyte(), DefaultMessageRoutine()

*This is the main program. It clears the message queue first, then go into the main loop program. In the main loop, it get all the messages from GetMessage(). It depends what message. It takes different actions. hibyte() and lobyte() for msg.lParam to get the high 8 bits and low bits.

*If program get some messages, it won't process. It will pass it to DefaultMessageRoutine(). It will pass the message to system message queue.

*/

```
static void wait_loop()
```

```
{
```

```
    U8 loopFlag=1;
```

```
    PIXEL px;
```

```
    GUI_MSG msg;
```

```
    short i=0;
```

```
    ClearMessageQueue();
```

```
    while (loopFlag) {
```

```
        GetMessage(sample,&msg);
```

```
        px.x=hibyte(msg.lParam);
```

```
        px.y=lobyte(msg.lParam);
```

```
        switch (msg.event) {
```

```
            case EVENT_CLICK:
```

```
                switch (msg.id) {
```

```
                    case MENUITEM_MAIN_EXIT:
```

```
                        loopFlag=0;
```

```
                        break;
```

```
                    case BUTTON_MAIN_PLAY:
```

```
                        /*sample action , BUTTON_MAIN_PLAY is pressed.
```

```
                        sample_play20() will send a message. It will add to the message queue
```

```
                        */
```

```
                        sample_play20();
```

```
                case MENUITEM_MAIN_EXIT:
```

```
                    /*EXIT*/
```

```
                default:DefaultMessageRoutine(sample,&msg);
```

```
                }
```

```
            break;
```

```
            default:DefaultMessageRoutine(sample,&msg);
```

```
        }
```

```
    }
```

```
}
```

U8 GetIndexFromId(FORM *frm, U8 id);

Get the sequential number (the location where it is defined in the form) of the component, whose identity is the second parameter.

Sample Code:

```
/*
 *The program demos the following APIs. GetIndexFromId( ) draw_rect(), MsgBox()
 *According id to get the index number of the Control , Use the index number to get parameters for
 a rectangle and using those parameters to draw a rectangle and display a message.
 */
void sample_play21(short id)
{
short idx;
GraphicsContext gc;
RECT temp;
idx=GetIndexFromId(sample,id);
temp.left=sample[idx].x;temp.right=sample[idx].x+sample[idx].w;
temp.top=sample[idx].y;temp.bottom=sample[idx].y+sample[idx].h;
gc.color = GUI_BLACK;gc.mode = Mode_SRC;
draw_rect(temp,&gc);
MsgBox("Do yuo see me?","sample",MB_OK);
};
```

U8 *GetScreenPtr(void);

Get the screen pointer. Yes, the screen pointer is useful for some functions such as bitblt();

Sample Code :

```
#define BITMAP_WIDTH 16
#define BITMAP_HEIGHT 11

const unsigned char pcBitmapData[] = {0xFF , 0xFF,
                                     0x80 , 0x01,
                                     0x8F , 0xF1,
                                     0x80 , 0x01,
```

```
0xFF , 0xFF});
```

```
void PaintBitmap(short sPositionX , short sPositionY)
{
    U8 * pcusScreen;

    GraphicsContext gc;

    gc.color = GUI_BLACK;
    gc.mode = Mode_SRC;
    gc.fill_index = BlackPattern;

    pcusScreen = GetScreenPtr();

    bitblt(0 , 0 , BITMAP_WIDTH , BITMAP_HEIGHT ,
           sPositionX , sPositionY , pcBitmapData ,
           BITMAP_WIDTH / 8 , pcusScreen , 20 , &gc);
}
```

U8 GetScreenHeight(void);

Get the screen height. Use this type of function can make a program more portable.

Sample Code: see PixelInRect()

U8 GetScreenWidth(void)

Get the screen width.

Sample Code: see PixelInRect()

Void SetAvailableRegion(RECT *r);

Specify some region in the screen to be available for some operations like drawing. For instance, you are going to draw in a special region, then you need use the function to specify the region.

Caution: Make sure that you have recover this setting by specify the region rectangle to the whole screen (now it is {0,0,159,159}) when finished, or it will affect other forms to display.

Sample Code:

```
/*
 *This program demos SetAvailableRegion() API.
 *Define a square. The center point is the center of the screen. Its length is 20 pixels.
 *Draw a lines from left hand upper corner to right hand button corner. Because the program reset
the screen available region, so it can draw the line within the available area.
 *Don't forget to to reset the screen size, when the program exit
 */
static sample_play28()
```

```
{
RECT temp;
temp.left=GetScreenWidth()/2-10;
temp.right=GetScreenWidth()/2+10;
temp.top=GetScreenHeight()/2-10;
temp.bottom=GetScreenHeight()/2+10;
SetAvailableRegion(&temp);
draw_line(0,0,GetScreenWidth(),GetScreenHeight(),GUI_BLACK);
temp.left=0;
temp.right=GetScreenWidth();
temp.top=0;
temp.bottom=GetScreenHeight();
SetAvailableRegion(&temp);
}
```

void draw_pixel (U16 x, U16 y, U8 color);

To draw a pixel in (x,y) by using some color.

Sample Code:

```
/*
*This program demos the following APIs - draw_pixel() , draw_line() and draw_ellipse().
*/
void sample_play23(PIXEL px)
{
    GraphicsContext gc;
    short idx;
    short x,y,x1,y1,x2,y2;
    gc.color = GUI_BLACK;gc.mode = Mode_SRC;gc.fill_index = BlackPattern;
    idx=GetIndexFromId(sample,BITMAP_MAIN);
    x=sample[idx].x+sample[idx].w/2;
    y=sample[idx].y+sample[idx].h/2;
    draw_pixel(x,y,gc.color);
    x1=sample[idx].x;
    y1=sample[idx].y;
    x2=sample[idx].x+sample[idx].w;
    y2=sample[idx].y;
    draw_line(x1,y1,x2,y2,gc.color);
    draw_ellipse(x,y,sample[idx].w/2,sample[idx].h/2,&gc);
}
```

void draw_line(short x1, short y1, short x2, short y2, short color);

Draw a line from pixel (x1, y1) to (x2, y2)

Sample Code: see void draw_pixel (U16 x, U16 y, U8 color);

void xor_draw_line(short x1, short y1 ,short x2, short y2);

Use the mode XOR to draw a line. It is often used in drawing. When the picture has background and you draw and modify some other graphics which may traverse existing picture, if you use function draw_line() to draw and erase the graphics, the background will be damaged. In this case, you can use the function to avoid this problem.

Use this function to draw_line is low efficient. I recommend not to use it as possible as you can.

Sample Code: see PixelInRect()

void draw_ellipse(short x,short y,short a ,short b,GraphicsContext *gc);

Draw an ellipse or a circle whose center point is (x, y). And “a” is the long axis, “b” is the short axis.

The member of “color” in *gc” is used as the color of the ellipse’s border, and member “fill_index” and “mode” is N/A.

Sample Code: see void draw_pixel (U16 x, U16 y, U8 color);

void bitblt(short src_x,short src_y,short w,short h,short dest_x,short dest_y,U8 *src,short src_units_per_line, U8 *dest,short dest_units_per_line,GraphicsContext *gc);

Remember the function bitblt in windows SDK? This function can copy, paste image between screen and memory, even if both the source and target bitmaps are in memory. In this function, member “color” of *gc is NOUSED!

Src_x,src_y,w,h: the region of source image
Dest_x,dest_y: the left-top corner of destination image
*src: Source image pointer
*dest: Destination image pointer
Src_units_per_line: To indicate how many units will be used in one line. For current LCD screen, this value must be 20, because the screen width is 160.
Dest_units_per_line: To indicate how many units will be used in one line. For current LCD screen, this value must be 20, because the screen width is 160.
*gc: The structure of the fill pattern and the copy method such as only simply paste the destination area, or the source image can be XOR the destination area while it paste. The structure GraphicsContext is defined in gui.h

```
typedef struct {
    Color color;
    CopyMode mode;
    PatternIndex fill_index;
} GraphicsContext;
```

```
typedef enum {
    GUI_WHITE =0,
    GUI_BLACK = 1
} Color;
```

```
typedef enum {
    Mode_SRC,
    Mode_NOT_SRC,
    Mode_SRC_OR_DST,
```

```
        Mode_SRC_AND_DST,
        Mode_SRC_XOR_DST,
        Mode_SRC_OR_NOT_DST,
        Mode_SRC_AND_NOT_DST,
        Mode_SRC_XOR_NOT_DST,
        Mode_NOT_SRC_OR_DST,
        Mode_NOT_SRC_AND_DST,
        Mode_NOT_SRC_XOR_DST,
        Mode_NOT_SRC_OR_NOT_DST,
        Mode_NOT_SRC_AND_NOT_DST,
        Mode_NOT_SRC_XOR_NOT_DST,
        InvalidMode
    } CopyMode;

    typedef enum {
        WhitePattern = 0,
        BlackPattern,
        DarkGreyPattern,
        LightGreyPattern,
        MicroPengPattern,
        InvalidPattern
    } PatternIndex;
```

Sample Code:

```
/*
 *This program demos the following APIs - bitblt() and PixelsUnits().
 *Show a bitmap image.
 */
void sample_play24()
{
    short idx;
    GraphicsContext gc;
    idx=GetIndexFromId(sample,BUTTON_SHOW_BITMAP);
    gc.color = GUI_BLACK;gc.mode = Mode_SRC;gc.fill_index = WhitePattern;
    bitblt(0,0,smilie_oops_bmp.width,smilie_oops_bmp.height,sample[idx].x+1,sample[idx].y+1,smilie_oops_bmp.img,PixelsUnits(smilie_oops_bmp.width),GetScreenPtr(),20,&gc);
}
```

U8 PixelsUnits(short pxs);

Get how many bytes a certain number pixels will be occupied. Pxs is the number of the pixels,and the function return the number of bytes.

Sample Code: see void bitblt(short src_x,short src_y,short w,short h,short dest_x,short dest_y,U8 *src,short src_units_per_line, U8 *dest,short dest_units_per_line,GraphicsContext *gc);

```
void patternfill(short dest_x,short dest_y, short w,short h,U8 *dest,
short dest_units_per_line,GraphicsContext *gc);
```

Use a pre-defined pattern to fill the given rectangular region, about structure *GraphicsContext* please see function `bitblt()`;

In `*gc`, member `color` is NOUSED this time. This function will use member `fill_index` as fill pattern and `mode` as fill mode. But strongly recommend to use function `FillRect()` instead of using this function. COZ function `FillRect` is power enough.

Sample Code :

```
void FillPatternDemo(void)
{
    GraphicsContext gc;

    U8 * pcusScreen;

    pcusScreen = GetScreenPtr();

    gc.mode = Mode_SRC;
    gc.fill_index = DarkGreyPattern;

    patternfill(16 , 120 , 24 , 24 , pcusScreen , GetScreenWidth() / 8 , &gc);
}
```

void FillRect(RECT r,GraphicsContext *gc);

Use the given GraphicsContext to fill a rectangle region. About GraphicsContext, please see function `bitblt()`;

Member `color` of `*gc` is NOUSED here.

Sample Code: see `PixelInRect()`

void draw_rect(RECT r,GraphicsContext *gc);

Only draw the outline of a rectangle region by using member `color` of `*gc`; member `fill_index` and `mode` is bypassed.

Sapmle Code: See `U8 GetIndexFromId(FORM *frm, U8 id)`;

void CopyScreen(U8 *p,U16 x,U16 y,U16 w,U16 h);

Copy the screen image to memory which start at 'p'

(x,y,w,h): The image region of screen

U8 *p: do NOT forget use some method such as `malloc()` to assign a memory block to pointer p. If you have malloced ,REMEMBER to free this block when you don not use it any longer.

This function is often used to save the screen display when you create another form. That's to say, save the background of the form, because it is not safe sometimes that you set a form's status to be `FORM_SAVE_BEHIND`. It often used with `CopyScreen()` to fulfill `save_behind`.

Sample Code: see `U8 *MallocScreen(U16 w,U16 h)`

void PasteScreen(U8 *p,U16 x,U16 y,U16 w,U16 h);

paste the contents of 'p' (maybe a bitmap) to screen.

(x,y,w,h): the image region of screen.

Sample Code: see U8 *MallocScreen(U16 w,U16 h)

U8 *MallocScreen(U16 w,U16 h);

According to the specified width & height to assign a memory block. Strongly recommend to use this function to assign memory if this memory block is for the screen operation purpose.

Sample Code:

```
/*
 *This program demos the following APIs - MallocScreen(), CopyScreen( ) CreateForm( ),
 CloseForm(), PasteScreen() and FreeScreen().
 *allocate a block of memory and save the current screen into this memory block
 *Create a new window and call the main loop.
 * Exit the program : close the windos, restore the screen and release the memory

*/
static void Sample(void)
{
    U8 *background;//momery block to save background
    background=MallocScreen(160,160);
    CopyScreen(background,0,0,160,160);
    CreateForm(sample);
    wait_loop();
    CloseForm(sample);
    PasteScreen(background,0,0,160,160);
    FreeScreen(background);
};
```

U8 FreeScreen(U8 *p);

Free memory block..

Sample Code: see U8 *MallocScreen(U16 w,U16 h)

void CreateForm(FORM *dlg);

Only display a form. For the details of *FORM* ,pls review 'FORM data structure'.

Sample Code: see U8 *MallocScreen(U16 w,U16 h)

void CloseForm(FORM *dlg);

close the current form.

Sample Code: see U8 *MallocScreen(U16 w,U16 h)

void HideComponent(short index,FORM *frm);

Let a component invisible. A componet can not fire any message When it is hidden. The first argument index is the sequential index of the component related to form caption. Second FORM *frm point to the address of the form the component in.

Sample Code:

```
/*
 *This program demos the following APIs - HideComponent() , ShowComponent( ).
 *Hide the object and display the hidden object.

 */
void sample_play26()
{
short idx;
idx=GetIndexFromId(sample,BUTTON_SHOW_BITMAP);
HideComponent(idx,sample);
Delay();
ShowComponent(idx,sample);
}
```

void ShowComponent(short index,FORM *frm);

Make a invisible component visible. The two arguments are as same as the two ones of HideComponent().

Sample Code : see void HideComponent(short index,FORM *frm);

void RefreshComponent(short index, FORM *frm);

When some component need to change while you do not want to create the form again,you can use this function. For instance, when click a button. a label in the form will be changed, you can change the string, the refresh the label. The two arguments are as same as the two ones of HideComponent().

Sample Code:

```
/*
 *This Program demos RefreshComponent() API.
 *When the attribute of object was changed, program should refresh the object
 *RefreshComponent() can refresh the object
 */
void sample_play25()
{
short idx;
```

```
        idx=GetIndexFromId(sample,BUTTON_SHOW_BITMAP);
        RefreshComponent(idx,sample);
    }
```

void SetSpinButtonText();

Sample Code:

```
/*
 *This program demos SetSpinButtonText() API.
 *Tap the arrow of SpinButton to change the text of SpinButton object( -99 到 99 )
 */
static sample_play27(short msgvalue)
{
    short i;
    i=atoi(spinbutton_sample.str)+msgvalue;
    if((i<100)&&(i>-100))
        SetSpinButtonText(&sample[GetIndexFromId(sample,SPINBUTTON_SAMPLE)],ito
        a(i));
}
```

RefreshMultiLineString(FORM *txt, short flag);

Refresh the content of the label. If flag is 1 , the text box is changed immediately, else the textbox is not changed and this function only for calculating the size of the label.

Remember the first argument 'txt' point to the FORM structure which represents the label, not the whole form.

Sample Code :

```
char szLabelCaption[32];
static FORM frmMainFrame[]=
{
    {FORM_CAPTION , 0 , 0 , 0 , SCREEN_WIDTH , SCREEN_HEIGHT ,
&cusFormCaption_MainFrame , FORM_FRAME_CAPTION , COMPONENT_VISIBLE ,
FONT_SMALL},

    {FORM_LABEL , LABEL_CONTENT_ID , 20 , 100 , 100 , 40 , szLabelCaption ,
X_ALIGN_LEFT | Y_ALIGN_CENTER | LABEL_BORDER | LABEL_MULTILINE ,
COMPONENT_VISIBLE , FONT_SMALL},

    {FORM_END},
};

void RefreshDemo(void)
{
```

```
int nIndex;

// According ID , to get form index ID for a label
nIndex = GetIndexFromId(frmMainFrame , LABEL_CONTENT_ID);

// Update Label attribute
sprintf(szLabelCaption , "Hello World\nLinuxDA");

// refresh the multi-line Label , and get the line number for Label object

RefreshMultiLineString(&frmMainFrame[nIndex] , 1);
}
```

Some fuctions about system register for applications:

short GetLaunchCommand(int argc,char *argv[]);

Ask for system to translate the parameters passed to the application.

Sample Code:see void init_screen(void)

void PutAppInfo(APP_INFO *app_info);

Register the application's information to the system

Sample code:(take address for example)

Add a resource file:**address.rsrc**. It is defined as follows:

```
#include <gui.h>
#define l_icon_width 21
#define l_icon_height 21
#define s_icon_width 14
#define s_icon_height 14
static unsigned char l_icon_bitmap[]={
    0x3F,0xFF,0xE0,0x7F,0xFC,0xF0,0xFF,0xF0,0x38,0xFF,0xE0,0x48,0xFF,0x80,0x88,
    0xFF,0x01,0x08,0xFE,0x04,0x08,0xFC,0x0E,0x18,0xF8,0x1E,0x38,0xF0,0x3F,0xF8,
    0xE0,0x7F,0xF8,0xE0,0xFF,0xF8,0xC0,0xFF,0xF8,0xC1,0x7F,0xF8,0xC2,0x7F,0xF8,
    0xC4,0x7F,0xF8,0xC8,0xFF,0xF8,0xD0,0xFF,0xF8,0xE3,0xFF,0xF8,0x7F,0xFF,0xF0,
    0x3F,0xFF,0xE0,0x00,0x00,0x00,0x41,0x64,0x6F,0x62,0x65,0x20,0x50,0x68,0x6F,
    0x74,0x6F,0x73,0x68,0x6F,0x70,0x20,0x44,0x6F,0x63,0x75,0x6D,0x65,0x6E,0x74,
    0x20,0x44,0x61,0x74,0x61,0x20,0x42,0x6C,0x6F,0x63,0x6B,0x00,0x00,0x00,0x00,
    0x00};
static unsigned char s_icon_bitmap[]={
    0x7F,0xF8,0xFF,0x8C,0xFE,0x04,0xFC,0x04,0xF8,0x04,0xF0,0xCC,0xE1,0xFC,0xC3,
    0xFC,0x87,0xFC,0x83,0xFC,0x93,0xFC,0xA7,0xFC,0xCF,0xFC,0x7F,0xF8,0x41,0x64,
    0x6F,0x62,0x65,0x20,0x50,0x68,0x6F,0x74,0x6F,0x73,0x68,0x6F,0x70,0x20,0x44,
```

```
0x6F, 0x63, 0x75, 0x6D, 0x65, 0x6E, 0x74, 0x20, 0x44, 0x61, 0x74, 0x61, 0x20, 0x42, 0x6C,
0x6F, 0x63, 0x6B, 0x00, 0x00, 0x00, 0x00};
static BITMAP large_icon={l_icon_width, l_icon_height, l_icon_bitmap};
static BITMAP small_icon={s_icon_width, s_icon_height, s_icon_bitmap};
APP_INFO app_info={
    "Address",
    "1.0.003",
    "LinuxDA Inc.",
    "LinuxDA Shanghai",
    &large_icon,
    &small_icon
};
```

app_info.h:

```
#ifndef _APP_INFO_H
#define _APP_INFO_H
typedef struct {
    char nick_name[16];
    char version[16];
    char owner[32];
    char creator[32];
    BITMAP *large_icon;
    BITMAP *small_icon;
}APP_INFO;
/***** Applications running mode *****/
#define CMD_RUN_NORMAL 1
#define CMD_RUN_GET_INFO 2
#endif
```

Modify the main function of address:

```
void main(int argc, char *argv[])
{
    short launch_cmd;
    init_screen();

    launch_cmd=GetLaunchCommand(argc, argv);
    switch(launch_cmd)
    {
        case CMD_RUN_NORMAL:
            OpenAddressDB();
            OpenIRDA();
```

```
        LoadAppStatus(&g_last_selected_item,&g_scr_list.position,&CardRec);
        InitAdresstCategory();
        GetListRegion();
        WaitLoop();
        CloseIRDA();
        LDACloseDB(g_db_handle, 0);
    break;
case CMD_RUN_GET_INFO:
    PutAppInfo(&app_info);
    break;
}
close_screen();
}
```

Some functions about textbox

void RefreshTextBoxContents(FORM *txt, short start_pos, short end_pos, short flag);

Refresh the content from position 'start_pos' to 'end_pos' in the textbox. If flag is 1 , the text box is changed immediately, else the textbox is not changed.

You may be puzzled for what means of the textbox not immediately refreshed. It is used in case that you only want to get the practical height of the textbox.This function will calculate the height of the textbox and give it to the textbox component.

Remember the first argument 'txt' point to the FORM structure which represents the textbox, not the whole form.

Sample Code :

```
TEXTBOX textbox_Content = {3 , "abcdefg\n12345678\nhijklmn" , 250 , 3 , 0 , 9};

static FORM frmMainFrame[] =
{
    {FORM_CAPTION , 0 , 0 , 0 , SCREEN_WIDTH , SCREEN_HEIGHT ,
    &cusFormCaption_MainFrame , FORM_FRAME_CAPTION , COMPONENT_VISIBLE ,
    FONT_SMALL},

    {FORM_TEXTBOX , TEXTBOX_CONTENT_ID , 20 , 20 , 100 , 60 , &textbox_Content , 0 ,
    COMPONENT_VISIBLE , FONT_SMALL},

    {FORM_END},
};

void RefreshDemo(void)
{
```

```
int nIndex;

// According ID to get the form index ID for TextBox
nIndex = GetIndexFromId(frmMainFrame , TEXTBOX_CONTENT_ID);

// Update attribute for TextBox
textbox_Content.string[0] = 'A';
textbox_Content.string[1] = 'B';
textbox_Content.string[2] = 'C';
textbox_Content.string[6] = 'Z';

// refresh the text box , and get the line number for TextBox object ( system use only)
// if the last paramater is 0, it won't refresh the object. It get the line number for TextBox only.

RefreshTextBoxContents(&frmMainFrame[nIndex] , 0 , 7 , 1);
}
```

void SetTextBoxText(FORM *txt, char *s);

Change the textbox content to the string which 's' point to and display it immediately. The argument 'txt' point to the FORM structure which represents the textbox, not the whole form.

Sample Code :

```
void SetTextBoxTextDemo(char szContent)
{
    int nIndex;
    //get ID, get form index id for TextBox

    nIndex = GetIndexFromId(frmMainFrame , TEXTBOX_CONTENT_ID);

    //Update Attribute for TextBox
    SetTextBoxText(&frmMainFrame[nIndex] , szContent);

    //Refresh TextBox
    RefreshComponent(nIndex , frmMainFrame);
}
```

short GetTextBoxLines(FORM *txt);

Return the total lines that the textbox has;

Remember the argument 'txt' point to the FORM structure which represents the textbox, not the whole form.

Sample Code :

```
void ShowTextBoxLineCount(U8 TextBoxID)
```

```
{
    int nIndex;

    int nTextBoxLineCount = 0;

    char szMessage[32];

    nIndex = GetIndexFromId(frmMainFrame , TextBoxID);

    nTextBoxLineCount = GetTextBoxLines();

    sprintf(szMessage , "TextBox Lines : %d " , nTextBoxLineCount);

    MsgBox(szMessage , "LinuxDA" , MB_OK);
}
```

void SetTextBoxStartLine(FORM *txt, short line);

Set the textbox start line as 'line', that means to change the parameter start_line in TEXTBOX structure;

Remember the argument 'txt' point to the FORM structure which represents the textbox, not the whole form.

Sample Code :

```
void ScrollTextBoxDemo(int nStartPosition)
{
    int nIndex;

    //get ID, get form index id for TextBox
    nIndex = GetIndexFromId(frmMainFrame , TEXTBOX_CONTENT_ID);

    // Update attribute for TextBox ,
    //It works when the actual line number in TextBox is bigger than nStartPosition.
    SetTextBoxStartLine(&frmMainFrame[nIndex] , nStartPosition);

    // Refresh TextBox
    RefreshComponent(nIndex , frmMainFrame);

    // After refresh TextBox, SetFocus is required
    SetFocus(frmMainFrame , TEXTBOX_CONTENT_ID);
}
```

void SetTextBoxEndLine(FORM *txt, short line);

Set the textbox end line as 'line', that means to change the parameter end_line in TEXTBOX

structure;

Remember the argument 'txt' point to the FORM structure which represents the textbox, not the whole form.

Sample Code :

```
void SetTextBoxVisibleLinesDemo(int nVisibleLines)
{
    int nIndex;

    //get ID, get form index id for TextBox
    nIndex = GetIndexFromId(frmMainFrame , TEXTBOX_CONTENT_ID);

    // Update TextBox Attribute
    SetTextBoxEndLine(&frmMainFrame[nIndex] , nVisibleLines);

    // Refresh TextBox
    RefreshComponent(nIndex , frmMainFrame);
}
```

void SetTextBoxEmptyLines(FORM *txt, short line);

Set the textbox minimum empty lines as 'line', that means to change the parameter min_empty_lines in TEXTBOX structure;

Remember the argument 'txt' point to the FORM structure which represents the textbox, not the whole form.

Sample Code :

```
void SetTextBoxEmptyLinesDemo(int nEmptyLines)
{
    int nIndex;

    //get ID, get form index id for TextBox
    nIndex = GetIndexFromId(frmMainFrame , TEXTBOX_CONTENT_ID);

    //Update attribute for TextBox.
    SetTextBoxEmptyLines(&frmMainFrame[nIndex] , nEmptyLines);

    // Refresh TextBox
    RefreshComponent(nIndex , frmMainFrame);
}
```

void SetFocus(FORM *frm, short id);

Set the component which id is 'id' in the FORM '*frm' as current focus. It is often used in the textbox;

Sample code: see `SetTextBoxtStartLine(FORM *txt, short line);`

U8 PixelInRect(PIXEL px,RECT r);

To check if a pixel in a rectangle region, if yes, return 1

Sample Code:

```
/*
 *This program demos the follwing APIs - GetScreenWidth() , GetScreenHeight() , PixelInRect().
 *xor_draw_line(). Define a square. Its center is in the center of Window. The length is 20 pixels
 *Check px within this square , If it is within the square, fill the square
 *If it is outside the square, draw a diagonal line.
 */
void sample_play22(PIXEL px)
{
    RECT temp;
    GraphicsContext gc;
    temp.left=GetScreenWidth()/2-10;
    temp.right=GetScreenWidth()/2+10;
    temp.top=GetScreenHeight()/2-10;
    temp.bottom=GetScreenHeight()/2+10;
    gc.color = GUI_BLACK;gc.mode = Mode_SRC;gc.fill_index = BlackPattern;
    if(PixelInRect(px,temp)==1)
        FillRect(temp,&gc);
    else xor_draw_line(temp.left,temp.top,temp.right,temp.bottom);
};
```

U16 GetBatteryPower(void);

Get the battery power,

Sample Code :

```
void ShowRestPower(void)
{
    int nCurrentPowerPercent;

    char szMessage[32];

    nCurrentPowerPercent = GetBatteryPower();

    sprintf(szMessage , "Current Power : %d %" , nCurrentPowerPercent);

    MsgBox(szMessage , "LinuxDA" , MB_OK);
}
```

void SetAutoOffTime(short time);

Set the waiting time before system automatic turn off the machine. 'time' is the number of the

secondes.

short GetAutoOffTime(void);

Get the waiting time before system automatic turn off the machine and return it

void SuspendMode(void);

Switch the system to suspend mode. It will save power if system is under this mode.

void PowerOff(void);

Turn off the system.

void SetContrast(short contrast);

Set the screen contrast. The contrast is expressed by a short number 'contrast'.

void SwitchBackLight(void);

Switch the status of the screen back –light. That is say, if the back-light is lid, turn it off. Otherwise turn it on.

Next three functions are used to control the volume of beep sound. The application programmer can give distinct sound for different situation. The volume of the beep is controlled by the system.

void SysBeep();

Give off a beep about system.

void AlarmBeep();

Give off a beep when it is time to alarm.

void GameBeep();

Give off a beep during the course of a game.

Some low level functions:

void OpenUART (void);

Open the UART. It is necessary to open UART before communicating with COM port.

void CloseUART (void);

Close the UART.

short ReadUART (U8 *data);

Read a byte from COM port, and put the byte in the parameter *data. Return 0 if success and return

-1 if timed out.

short WriteUART (U8 code);

Transit a byte *code* to COM port. Return 0 if success, and return -1 if timed out.

IRDA functions:

void OpenIRDA(void);

Open the IRDA. You must open the IRDA before communicating with other PDAs.

void CloseIRDA(void);

Close the IRDA.

short ReadIRDA(void *data,long size);

Read data from the IRDA port according to the size you want. Return the size if success, return -1 if timed out.

short WriteIRDA(void *data,long size);

Send data to the IRDA port according to the size you want. Return the size if success, return -1 if timed out.

Sample code:

Suggestion: Call OpenIRDA() at the beginning of your main function and CloseIRDA() at the end of it.

```
void sendbyIrda(ADDRESS_T *rec)
{
    short state,sendtry=0;
    background=(U8*)MallocScreen(160,160);
    CopyScreen(background,0,0,160,160);
    CreateForm(frmBeam);
    state=WriteIRDA(rec,sizeof(ADDRESS_T));
    while(state<0){
        sendtry++;
        if(sendtry>5)
            break;
        state=WriteIRDA(rec,sizeof(ADDRESS_T));
    }

    if(state<0){
        MsgBox(IRDA_SEND_FAILS,FRM_IRDA_SEND,MB_OK);
    }
    else {
        MsgBox(IRDA_SEND_SUCCESS,FRM_IRDA_SEND,MB_OK);
    }
    CloseForm(frmBeam);
    PasteScreen(background,0,0,160,160);
    FreeScreen(background);
}
```

```
}
```

```
void recvbyIrda(ADDRESS_T *RecvRec)
{
    short status;
    status=ReadIRDA(RecvRec,sizeof(ADDRESS_T));
    if(status==sizeof(ADDRESS_T)){
        strcpy(message,"");
        strcat(message,RecvRec->last_name);
        strcat(message,RecvRec->first_name);
        answer= MsgBox(message,FRM_IRDA_RECEIVE,MB_YESNO);
        if (answer==MB_YES)
            LDAAddRecord(g_db_handle,RecvRec,0,0);
    }
}
```

short GetRegister (U32 reg, void *data, short size);

Get the data in the specified register. The parameter 'reg' represents the register and the 'data' is the data in this register. The parameter 'size' can be 1, 2, or 4 that means the data type is char, short or int(or long). This function will return 0 if success and -1 if fail.

If the size is 2, the parameter 'reg' must be the even address, otherwise the function will return -1; And if the size is 4, 'reg' must be the multiple of 4.

For MOTOROLA Dragon Ball serial chipsets, the register is the image of the memory address, and this function can read memory directly.

short SetRegister (U32 reg, void *data, short size);

Set the data in the specified register. The parameter 'reg' represents the register and the 'data' is the data in this register. The parameter 'size' can be 1, 2, or 4 that means the data type is char, short or int(or long). This function will return 0 if success and -1 if fail.

For MOTOROLA Dragon Ball serial chipsets, the register is the image of the memory address, and this function can read memory directly.

U32 SetCPUSpeed(U8 p, U8 q, U8 freq);

U32 GetCPUSpeed(U8 p, U8 q, U8 freq);

void DisableSysHardwarekb(U8 kc);

Disable a hardware key. And there is no message given when the key is pressed. The parameter kc is the keycode of the key. Remember to use the next EnableSysHardwarekb() to reuse the key.

If you do not want to use this function for you may forgot to recover it. There is another method. Put a case sentence without processing instructions other than 'break' into the switch one. Therefore the message, whose keycode is as same as the one follows the 'case', will be neglected.

void EnableSysHardwarekb(U8 kc);

Enable a hardware key after it is disabled.

In "msgbox.h"

short MsgBox(char * msgstr, char *title, short mode);

This function can display a form which shows a message in the center. The form is made up of a caption shown in the left top corner, a message string, and one or two buttons. These contents are controlled by the three parameters.

*msgstr ----- The message shown in the center of the form.

*title ----- The caption.

mode ----- what type the buttons is, which is used to close form and return a value. There are three styles you can choose from: MB_OKCANCEL, MB_YESNO, MB_OK. Each type of buttons the mode defined as follows:

MB_OKCANCEL:

MB_YESNO:

MB_OK:

The function return a value decided by which button the user clicks. It can be MB_OK, MB_CANCEL, MB_YES, MB_NO.

This form neglects the key RETURN.

Sample Code :

```
int g_nModified;

void SaveConfirm(void)
{
    short sAnswer;

    if(g_nModified)
    {
        sAnswer = MsgBox("Do you want to Save it?" , "LinuxDA" , MB_YESNO);
        switch(sAnswer)
        {
            case MB_YES:
                SaveProcess();
                CloseProcess();
```

```
        break;
    case MB_NO:
        CloseProcess();
        break;
    }
}
else
{
    CloseProcess();
}
}
```

In "get_msg.h"

void GetMessage(FORM *frm,GUI_MSG *msg);

YES, It's the most important function to get the message queue!!

Frm	The pointer of the form who wants to get the message
Msg	The return message, for more details , pls see the next chapter

Sample Code: see void ClearMessageQueue(void);

Void DefaultMessageRoutine(FORM * dlg, GUI_MSG *msg);

This is a very important function to process the message. Any message will not processed in you're application should be sent to the system by this function and those caught by your application should not be sent back. If you perplex with it, refer to the introduction of the message queue and the simple example in the back of this manual.

Sample Code: see void ClearMessageQueue(void);

Void SendMessage(GUI_MSG *msg);

Sample Code:

```
/*
 *This program demo SendMessage( ) API.
 * It shows you how to send the message。 It emulates CLICK event in menu.

*/
void sample_play20()
{
    GUI_MSG exit_msg;
    exit_msg.event=EVENT_CLICK;
    exit_msg.id=MENUITEM_MAIN_EXIT;
    SendMessage(&exit_msg);
}
```

```
};
```

```
U8 lobyte(U16 IParam);
```

Sample Code :

```
int EventClick_Process(GUI_MSG * pcusMessage)
{
    int nStickPositionX;
    int nStickPositionY;

    char szMessage[32];

    int nResult = pcusMessage->id;

    nStickPositionX = hbyte(pcusMessage->IParam);
    nStickPositionY = lobyte(pcusMessage->IParam);

    sprintf(szMessage , "Current Click Position: X - %d , Y = %d" ,
            nStickPositionX , nStickPositionY);

    MsgBox(szMessage , "LinuxDA" , MB_OK);

    return nResult;
}
```

```
U8 hbyte(U16 IParam);
```

Get the high byte or the low byte of a 16 bit value

Sample Code : U8 lobyte(U16 IParam);

In "fonts.c"

```
U16 GetStringWidth(char *str);
```

Return the current font width

Sample Code :

```
#define MAP_WIDTH  150
#define MAP_HEIGHT 110

void ShowWelcome(char * szWelcomeInfo)
{
    int nPositionX;
    int nPositionY;

    GraphicsContext gc;

    // Display the Welcome Information
```

```
gc.color = GUI_BLACK;
gc.mode = Mode_SRC;
gc.fill_index = WhitePattern;

nPositionX = (MAP_WIDTH - GetStringWidth(szWelcomeInfo , FONT_SMALL)) / 2;
nPositionY = MAP_HEIGHT / 2;

do_drawString(nPositionX , nPositionY , szWelcomeInfo , &gc , FONT_SMALL);
}
```

U16 GetFontHeight(void);

Return the current font height

Sample Code :

```
static void ShowFontInfo(short nIndex)
{
    short sFontHeight;

    char szMessage[32];

    SetFont(FONT_SYS);

    sFontHeight = GetFontHeight();

    sprintf(szMessage , "System Font Height : %d " , sFontHeight);

    MsgBox(szMessage , "LinuxDA" , MB_OK);
}
```

void SetFont(Short font_lib)

Sets default font .Now it can only be set to 1

Sample Code : see U16 GetFontHeight(void);

void do_drawString(int x, int y, char *data,GraphicsContext *gc, short font_lib) ;

Display the specified string in the given location and given fonts. That means you can choose the font of the string by the last parameter, and the default font library is set at the same time.

Sample Code : see U16 GetStringWidth (char *str);

void DrawStringInRect(RECT *r, char *str, GraphicsContext *gc, short font_lib, short align_flag)

Try to display a string in a rectangle area; if the specified string is out of the given region, it will be clip to fit in the rectangle area.

Align_flag can set how the string is aligned in the rectangle.

Sample Code :

```
void DisplaySomething(char szMessage)
{
    RECT rectRange;

    GraphicsContext gc;

    // set up the area
    rectRange.top = 10;
    rectRange.left = 10;

    rectRange.bottom = rectRange.top + 100;
    rectRange.right = rectRange.left + 100;

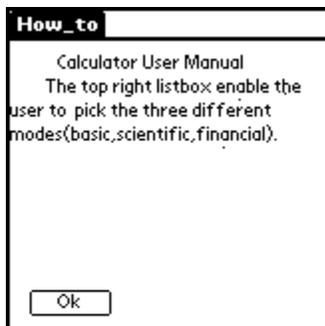
    // set up some attributes
    gc.mode = Mode_SRC;
    gc.fill_index = WhitePattern;
    gc.color = GUI_BLACK;

    // call the function to display the string
    DrawStringInRect(&rectRange ,
                    szMessage ,
                    &gc ,
                    FONT_SMALL ,
                    Y_ALIGN_CENTER | X_ALIGN_MIDDLE);
}
```

short gui_how_to(char *str);

This functions can display an application's help. The parameter “*str” is the text content of the application. It will show a form as Figure 1.4.1. You need to control the format of the text in the string you transfer to this function.

This function return the keycode of “RETURN” if you exit the form by clicking “RETURN” key.



Sample Code :

```
#define STRING_HOWTO_CONTENT " Snake :
- The idea of the game is change the direction of the snake without
its body touching the sides
- The settings menu allows the user to adjust the speed, stage, and
```

define the buttons

- The buttons that can be defined are the Schedule key, the Address key, the Up and Down Scroll keys , the ToDo list and the Memo Pad keys."

```
int FunctionProcess(GUI_MSG cusMessage)
{
    int nResult = cusMessage.id;

    switch(cusMessage.id)
    {
        case MENU_EXIT_ID:
            CloseForm(frmMainFrame);
            break;
        case MENU_HOWTO_ID:
            gui_how_to(String_Howto_Content);
            break;
    }

    return nResult;
}
```

short gui_about(char *title, char *version);

This functions can display an application's about. The parameter “*title” is application name. And the “*version” is the version string in format “v.x.x.xxx”. It will show a form as Figure 1.4.2 .

This function return the keycode of “ RETURN ” if you exit the form by clicking “RETURN” key.



Figure 1.4.2

Sample Code :

```
#define STRING_ABOUT_VERSION    "0.9.8.0"
```

```
int FunctionProcess(GUI_MSG cusMessage)
{
    int nResult = cusMessage.id;

    switch(cusMessage.id)
    {
        case MENU_EXIT_ID:
            CloseForm(frmMainFrame);
            break;
        case MENU_ABOUT_ID:
            gui_about(String_About_VerSiion);
            break;
    }

    return nResult;
}
```

5. GUI Internal Functions

These functions are declared as static functions. They will ONLY invoked by other GUI functions. That means applications can NOT call these functions directly.

In “*mini_gui.c*”

inline void draw_pixel(U16 x, U16 y, U8 color);

Use the specified color to draw a specified pixel. Consider of the speed, I define the function as a ‘inline’ function. This method may increase the invoke speed but will use more memory. So, strongly recommend **NOT** to invoke this function directly from the application.

inline void draw_xish_line(U16 x, U16 y, U16 dx, U16 dy, U16 xdir, U8 color);

inline void draw_yish_line(U16 x, U16 y, U16 dx, U16 dy, U16 xdir, U8 color);

these two function is the sub-routine of function draw_line.

static void draw_bitmap(FORM *mnu);

Maybe it’s only for the test reason. I only use this function to test the video frame buffer during the early period of my developing time. ***DON’T use this function !***

static void draw_menu(FORM *mnu);

According to the resource header to draw the menu. The calculation of the location of the menu

will be done in this routine.

static void draw_form_frame(RECT r,char *str,U8 style);

To draw the form border and the caption

static draw_button(short x,short y,short w,short h,char *str);

To draw the standard button of my GUI.

in "get_msg.c"

static U8 GetSelectedComponentId(PIXEL px,FORM *dlg);

It's the most complex function in the whole GUI. This function will generate the high-level message queue and do the default actions of standard components, such as the sub menu item will drop down while click the menu, reverse the face of the button once pen-down and restore the button face while pen-up.

short SendKeyToForm(FORM *frm,short code)

FORM* :the pointer of the whole form structure.

Code :ASCII code of the character.

Sample Code :

```
void AddCharToTextBox(int nCharCode)
{
    // NULL means the current Form , nCharCode is its corresponding ASCII code
    SendKeyToForm(NULL , nCharCode);

    // Can send key to other form
    // SendKeyToForm(frmMainFrame , nCharCode);
}
```

short GetCurrentActivatedTextBox(void)

Return the current active textbox ID. Return 0 if there is no selected id.

Sample Code :

```
void ShowTextBoxID(void)
{
    int nCurrentTextBoxID;

    char szMessage[32];
```

```
nCurrentTextBoxID = GetCurrentActivatedTextBox();

if(nCurrentTextBoxID)
    sprintf(szMessage , "Current Activated Text Box ID : %d %" , nCurrentTextBoxID);
else
    sprintf(szMessage , "No TextBox Activated!");

MsgBox(szMessage , "LinuxDA" , MB_OK);
}
```

static void pull_up_menu(FORM *dlg)

If there is some sub menu item is dropped, while click in any other components or click in a empty area, the menu will pull-up automatically.

Function "GetSelectedComponentId" will invoke it.

static void drop_down_menu(FORM *mnu);

Auto drop down the menu.

Function "GetSelectedComponentId" will invoke it.

static void GetMenuRect(RECT *r,FORM *mnu)

return the rectangular region of the menu

void XorButtonFace(RECT r);

Use this function to inverse the rectangular region

Function "GetSelectedComponentId" will invoke it.

Sample Code :

```
RECT g_rectPreviousHighLight;
static void HighLightApplication(U8 cusComponentID)
{
    int nIndex;
    RECT rectLabel;

    // According ID to get the form index ID
    nIndex = GetIndexFromId(&frmAppCategory , cusComponentID);

    // According to index to get location of Label
    rectLabel.left = frmAppCategory[nIndex].x - 1;
```

```
rectLabel.top = frmAppCategory[nIndex].y;
rectLabel.right = rectLabel.left + frmAppCategory[nIndex].w;
rectLabel.bottom = rectLabel.top + frmAppCategory[nIndex].h;

//reverse the color. Black to white. White to blank.
XorButtonFace(rectLabel);

// store the location for highlight Label
g_rectPreviousHighLight = rectLabel;
}
static short max(short x1,short x2);
hehe... :-P return the bigger value

static U8 LoopTillPenUp(RECT r);
A empty message loop, but I fell this method is so mesh, I will increase it soon.

static U16 isSystemRegion(PIXEL px);
To check if the pixel in a system region

Function "GetSelectedComponentId" will invoke it.
```

6. Message Queue Definitions and Related Function

Events definitions: in 'gui.h'

```
/******
*
* I define the events of internal message queue here
*
* Mac Tseng 11/2/2000
*
*****/
#define EVENT_NOTHING 0
#define EVENT_CLICK 1
#define EVENT_INPUTCHAR 2
#define EVENT_KEYDOWN 3
#define EVENT_KEYUP 4
#define EVENT_PEN_UP 5
#define EVENT_DESTROY_FORM 6
```

```
#define EVENT_SHOW_MENUBAR 7
```

```
#define EVENT_HIDE_MENUBAR 8
```

Event type Descriptions

Event type	Descriptions
EVENT_NOTHING	No events.
EVENT_CLICK	This event will be sent to the message queue ceaselessly when the pen click the touch screen until the pen leaves the screen. Caution: A clicking that you feel may cause a couple of this type events.
EVENT_INPUTCHAR	The event happens when you leave the Keyboard form(or Magic Pen form) for inputting characters into a textbox.
EVENT_KEYDOWN	It is the event caused by pressing the hardware button “▼”.
EVENT_KEYUP	It is the event caused by pressing the hardware button “▲”.
EVENT_PEN_UP	The event happens when hen the pen leave the touch screen.
EVENT_DESTROY_FORM	When the form is destroyed;
EVENT_SHOW_MENUBAR	The event when the menubar is shown.
EVENT_HIDE_MENUBAR_	The event when the menubar is hidden.

I separate the message queue into two parts, one is the low level, and the other is hi level.

Consider the response time period of pen interrupt. I don't want the interrupt program to get too much CPU's time. So, I DO simply queue the event CLICK and (x,y) when the pen interrupt occurred.

Most of the applications only care about high-level message queue. At least I do think so. Why a application needs to use the low level message queue? Because the low level message queue only returns a simple structure, it's not enough to deal with the events of UI (user interface).

The mostly job of message queue is done by GUI. So, I call this as 'high level message queue'

the low-level message queue

```
typedef struct {  
    U16 x;  
    U16 y;  
    U16 event;  
}S_MESSAGE;
```

the high level message queue

```
typedef struct {  
    U8 id;  
    U16 event; /* the type of the event*/  
    U32 IParam; /* a long parameter which can return different value according to
```

```
        different conditions, such as coordinates.*/
U8 keycode; /* the value returned when you press some hardware button on the shell
             of your PDA machine. Next you will find a list definitions that indicates
             what will be returned to every key on PDA.

short value; /* It has different use to different components or events. For
             SPINBUTTON, it will be -1 or 1 when the up or down arrow is clicked;
             for EVENT_INPUTCHAR, it will be the amount of lines discrepancy
             about the textbox.*/

}GUI_MSG;
```

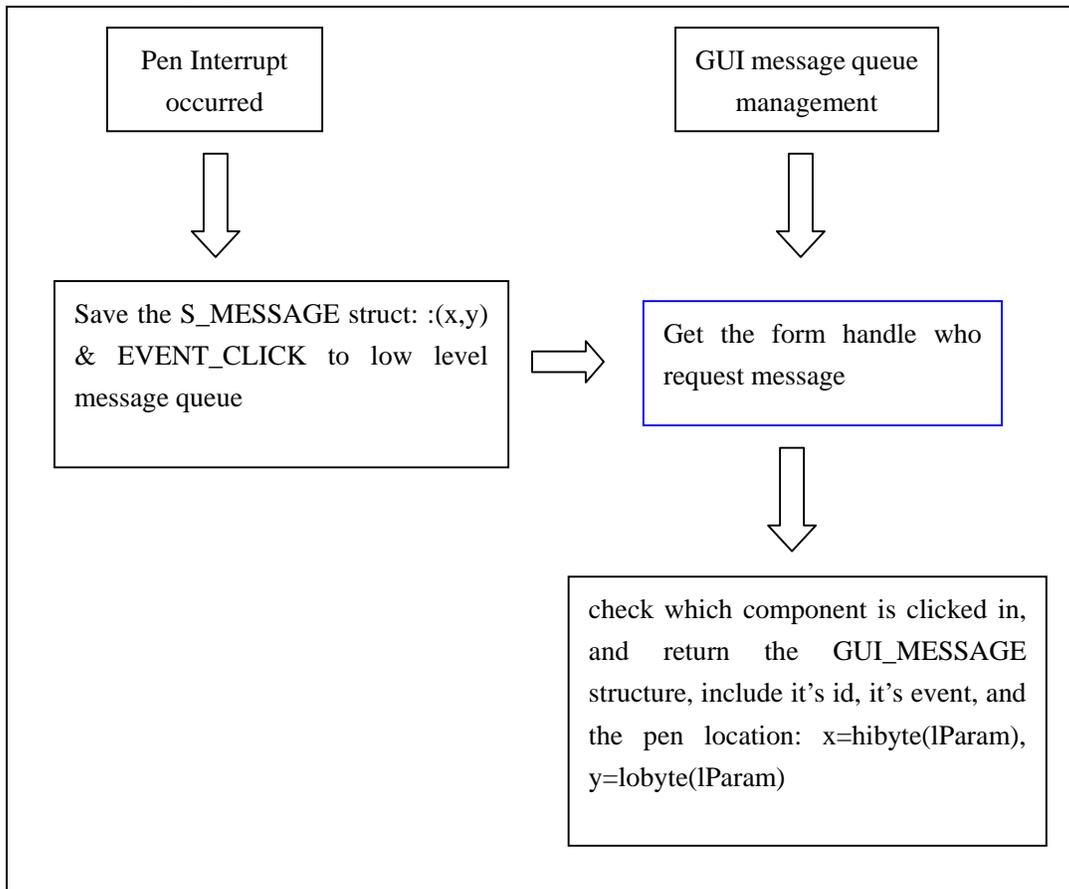
The Procomp PDA keycode

There are five hardware buttons on the Cellvic machine ,and the keycode of these buttons are defined as below:

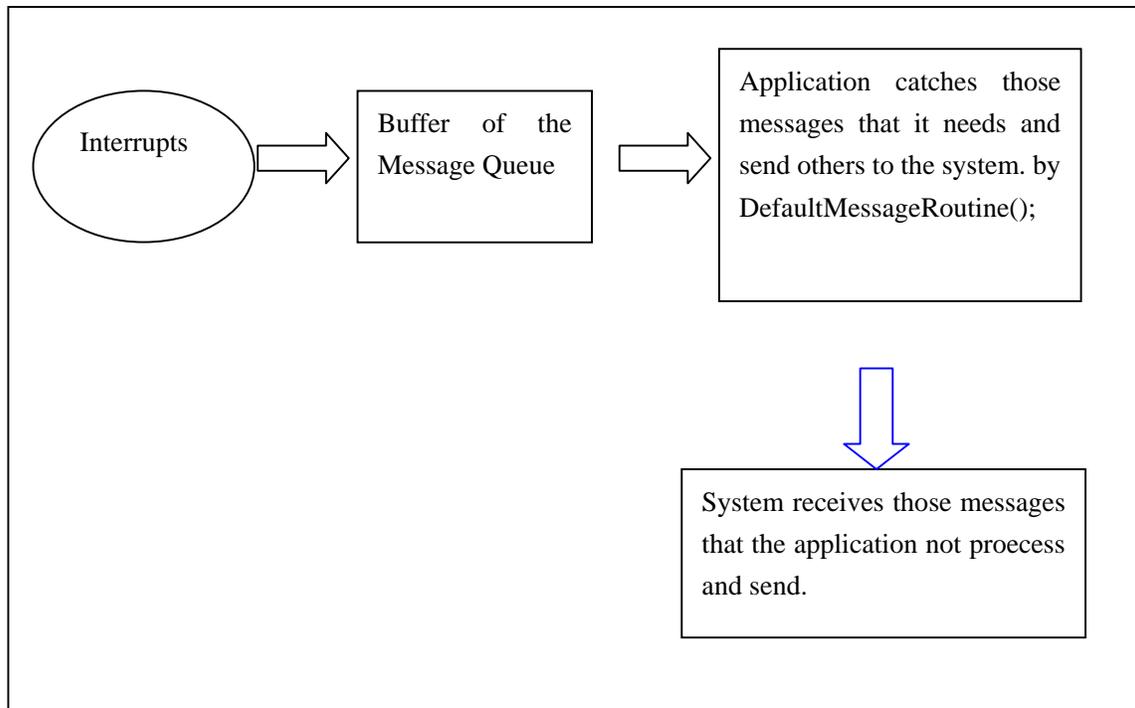
```
#define KEY_POWER          1
#define KEY_RIGHT          2
#define KEY_UP             3
#define KEY_DOWN          4
#define KEY_ENTER         5
#define KEY_LEFT          6
#define KEY_RETURN        7
#define KEY_KEYBOARD      8
#define KEY_LIGHT         9
#define KEY_ADDRESS       10
#define KEY_CALCULATOR   11
#define KEY_CLOCK         12

#define MAX_KEYMSG_COUNT 16
```

The Whole message queue flow chart



The flow chart of the processing message queue:



Simple Example

Here is a simple to show how to get the message and deal with it with the UI.

Pls notice:

1. it's very easy to create a application and u needn't to use any function such as `draw_line()`, `draw_rect()` to deal with the UI (User Interface).
2. Don't forget to get the message by using function `GetMessage()`;
3. Don't forget to send the messages you do not process to the system by function `DefaultMessageRoutine()`, or some errors will be caused.
4. Yes! It's only the easy standard components can be use in this sample, but, if u wanna create ur own one, why not try to use component 'BITMAP' ?

```
#include <stdio.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/mman.h>
#include <sys/types.h>
#include "linux/gui.h"
```

```
#include "mini_gui.h"

static void wait_loop(void);

FORMCAPION form_caption={"About US"};
static FORM frmAbout[]={
    {FORM_CAPTION ,0, 0, 0, 160, 160, &form_caption, 0, COMPONENT_VISIBLE,
FONT_MEDIUM},
    {FORM_MENU, 10, 5, 20, 0, 0, "Tools", 0, COMPONENT_VISIBLE, FONT_MEDIUM },
    {FORM_MENUITEM, 11, 0, 0, 0, 0, "Digitizer", 0, 0, FONT_MEDIUM },
    {FORM_MENUITEM, 12, 0, 0, 0, 0, "About us", 0, 0, FONT_MEDIUM },

    {FORM_MENU,20,45,20, 0, 0, "ABCD", 0, COMPONENT_VISIBLE, FONT_MEDIUM },
    {FORM_MENUITEM,21,0,0,0,0,"Digitizer",0,FONT_MEDIUM },
    {FORM_MENUITEM,22,0,0,0,0,"About us",0,FONT_MEDIUM },
    {FORM_MENUITEM,23,0,0,0,0,"Hi every body",0,FONT_MEDIUM },

    {FORM_MENU,30,90,20,0,0,"LinuxDA",0,COMPONENT_VISIBLE,FONT_MEDIUM },
    {FORM_MENUITEM,31,0,0,0,0,"Digitizer",0,FONT_MEDIUM },
    {FORM_MENUITEM,32,0,0,0,0,"About us",0,FONT_MEDIUM },
    {FORM_MENUITEM,33,0,0,0,0,"Hi every body",0,FONT_MEDIUM },

    {FORM_LABEL,0,8,35,60,60,"CEO: Paul Luang", X_ALIGN_LEFT|Y_ALIGN_MIDDLE ,
COMPONENT_VISIBLE, FONT_SMALL},
    {FORM_LABEL,0,8,50,60,60,"COO: Kiffin Tam", X_ALIGN_LEFT|Y_ALIGN_MIDDLE,
COMPONENT_VISIBLE, FONT_SMALL },
    {FORM_LABEL,0,8,65,60,60,"-----", X_ALIGN_LEFT|Y_ALIGN_MIDDLE,
COMPONENT_VISIBLE,FONT_SMALL },

    {FORM_BUTTON,50,70,90,35,20,"Click!",0,COMPONENT_VISIBLE,FONT_SMALL },
    {FORM_BUTTON,1,70,130,45,20,"Accept",0,COMPONENT_VISIBLE,FONT_SMALL },
    {FORM_END},
};

void Main(void)
{
    CreateForm(frmAbout);
    wait_loop();
}

static void wait_loop(void)
```

```
{
    U8 loopFlag=1;
    int i;
    GUI_MSG msg;
    PIXEL px;

    while (loopFlag) {
        GetMessage(frmAbout,&msg);
        switch (msg.event) {
            case EVENT_CLICK:
                switch(msg.id){
                    case 1: // click the button ' Accept' and then close form and go out.
                        printf ("out of about\n");
                        CloseForm(frmAbout);
                        LoopFlag = 0;
                    break;
                    default:
                        /*All those clicking message not processed should be sent to system*/
                        DefaultMessageRoutine(frmAbout,&msg);
                    break;
                }
            break;
            default:
                /*All other type events not processed should be sent to system*/
                DefaultMessageRoutine(frmAbout,&msg);
            break;
        }
    }
}
```

7. Database API

This section introduces the database data definitions and functions:

In “*mini_db.h*”

```
typedef int    DBHANDLE;
/*****
 *
 *   General marcos of DB
 *
 *****/
#define MAX_FILENAME_LENGTH    16

/*****
 *
 *   Definitions of field type of DB
 *
 *****/
#define DBCHAR    0x0001
#define DBINT     0x0002
#define DBLONG   0x0003
#define DBDATE   0x0004
#define DBTIME   0x0005
#define DBSTRING 0x0006
#define DBBINARY 0x0007
#define DBVARCHAR 0x0008

/*****
 *
 *   Definitions of record property of DB
 *
 *****/
#define DB_RECORD_BUSY    0x01
#define DB_RECORD_SECURITY 0x02
#define DB_RECORD_DELETED 0x04
#define DB_RECORD_UPDATED 0x08
#define DB_CATEGORY_ALL   0x10
#define DB_SHOW_SECURITY 0x00
```

```
#define DB_HIDE_SECURITY 0x02

/*****
 *
 *   Definitions of Error Numbers of DB API
 *
 *****/

#define DB_ERROR_OPENDATABASE      -1
#define DB_ERROR_CLOSEDATABASE    -1
#define DB_ERROR_READDATABASE     1002
#define DB_ERROR_WRITEDATABASE    1003
#define DB_ERROR_CREATEDATABASE   1004
#define DB_ERROR_DELETEDATABASE   1005
#define DB_ERROR_NOERROR          0
#define DB_ERROR_ADDCATEGORY      1006
#define DB_ERROR_DELETECATEGORY   1007
#define DB_ERROR_UPDATECATEGORY   1008
#define DB_ERROR_GETCATEGORY      1009
#define DB_ERROR_SETDBCATEGORY    1010
#define DB_ERROR_ALLOCATEMEMORY   1011
#define DB_ERROR_ADDRECORD        1012
#define DB_ERROR_DELETERECORD     1013
#define DB_ERROR_UPDATERECORD     1014
#define DB_ERROR_GETRECORD        1015

/*****
 *
 *   CreateDatabase flags
 *
 *****/

#define DB_SET_OVERWRITE 0x10
// #define DB_SET_NO_FIELDHEAD 0x20
#define DB_BLOCK_NONE    0
#define DB_BLOCK_16      4
#define DB_BLOCK_32      5
#define DB_BLOCK_64      6
#define DB_BLOCK_128     7
#define DB_BLOCK_256     8
#define DB_BLOCK_512     9
```

```
#define DB_BLOCK_1024    10
```

```
typedef struct{
    char field_name[MAX_FILENAME_LENGTH];
    shortfield_length;
    shortfield_type;
}DBFIELD;
```

```
/******
```

```
*
```

```
*   Open Database flags
```

```
*
```

```
*****/
```

```
#define DB_READ_ONLY    1
```

```
#define DB_EXCLUSIVE    2
```

```
//#define DB_APPEND_DATA  4
```

```
#define DB_PURGE_DATA   5
```

```
/******
```

```
*
```

```
*   Functions of DB
```

```
*
```

```
*****/
```

```
/******
```

```
*
```

```
*   Create a new database
```

```
*
```

```
*   Return Value: -1:      Create failed
```

```
*                   none zero:   handle of the table
```

```
*
```

```
*****/
```

```
int LDACreateDB(char *DBName,short DBFlags,DBFIELD
*dbField,short dbFieldNumber);
```

Create a new database. Parameter 1 is the database name. You can turn to the macro definition for DBFlags. Pointer dbField point to a DBFIELD structure which save the information about the database , and dbField Number is the count of the fields in the db.

Sample Code:

```
static DBFIELD g_db_field_detail[] = {
    {"last_name",25,DBSTRING},
    {"first_name",25,DBSTRING},
};
#include <mini_db.h>
static void OpenAddressDB(void)
{
    int g_db_handle ;
    g_db_handle = LDAOpenDB("address",DB_READ_ONLY);
    if( g_db_handle == -1) // fail to open database
    {
        LDACreateDB("address",DB_SET_OVERWRITE,g_db_field_detail,sizeof(g_db_field
_detail)/sizeof(DBFIELD));
        g_db_handle = LDAOpenDB("address",DB_READ_ONLY);
        LDAAddDBCategory(g_db_handle,"Unfiled");//add initial category
        LDAAddDBCategory(g_db_handle,"Personal");
    }
    LDACloseDB(g_db_handle,DB_READ_ONLY);
}
```

int LDAAddDBCategory(DBHANDLE fdidx,char * cate_name);

Add a category to the database. The function returns the category ID in the database. Category is a especial field named “Category” and defined by system, and you need not define it in DBFIELD. You can edit category by API such as add , delete, update a category. Every record in DB belongs to a category. A database can hold 16 kinds of category.

Sample Code: see `int LDACreateDB(char *DBName,short DBFlags,DBFIELD *dbField,short dbFieldNumber);`

int LDADeleteDBCategory(DBHANDLE fdidx,int cate_index);

Delete a category of the database which handle is fdidx. If a category existing in the database is deleted, the category all records belonging to it will be updated to 0 by system.

Sample Code:

```
#include <mini_db.h>
static void DeleteCategory(void)
{
    int g_db_handle ;
    int cate_index;

    g_db_handle = LDAOpenDB("address",0);
    if( g_db_handle != -1) // open database sucessfully
```

```
    {
        cate_index=LDAAddDBCATEGORY(g_db_handle,"Personal");//Record the Category ID
        of "personal"
        LDADeleteDBCATEGORY(g_db_handle,cate_index);
        LDACloseDB(g_db_handle,0);
    }
}
```

int LDAUpdateDBCATEGORY(DBHANDLE fdidx, int cate_index, char * cate_name);

Update a category name. Cate_index is the ID of the category.

Sample Code:

```
#include <mini_db.h>
static void UpdateCategory(void)
{
    int g_db_handle ;
    int cate_index;
    g_db_handle = LDAOpenDB("address",0);
    if( g_db_handle != -1 ) // open database successfully

    {
        cate_index=LDAAddDBCATEGORY(g_db_handle,"Personal");//Record the Category ID
        of "personal"
        LDAUpdateDBCATEGORY(g_db_handle,cate_index,"Business");
        LDACloseDB(g_db_handle,0);
    }
}
```

int LDAGetDBCATEGORYTotal(DBHANDLE fdidx);

Get the total number of categories in the database which handle is fdidx .

Sample Code: see static void GetCategoryNameStr(void)

int LDAGetDBCATEGORYAlphabet(DBHANDLE fdidx,int cate_rank, char * cate_name);

Get the category ID from the return value of this function. Pls pay attention to cate_rank. Cate_rank is not the ID of the category but the sequence index which is determined by sorting alphabetically. And the string pointer cate_name is the name of the category.

This function is very important to database programming if your database has categories. You can get IDs and names of all categories by this function.

Sample Code:

```
static void GetCategoryNameStr(void)
{
    short i;
```

```
int g_db_handle ;

int g_cate_total;
char g_cate_id[16];//store the catogory IDs
char g_cate_str[16][20];//store the category names
g_db_handle = LDAOpenDB("address",0);
if( g_db_handle != -1) // open database sucessfully

{
    g_cate_total = LDAGetDBCATEGORYTotal(g_db_handle);//Get the total numbers of
category
    for(i = 0;i < g_cate_total;i++){
        g_cate_id[i] = LDAGetDBCATEGORYAlphabet(db_handle,i,g_cate_str[i]);
    }
    LDACloseDB(g_db_handle,0);
}
}
```

DBHANDLE LDAOpenDB(char *DBName,short DBFlag);

Open a database whose name is DBName. You can set DBFlag referring to the macro definitions.

Sample Code: see int LDACreateDB(char *DBName,short DBFlags,DBFIELD *dbField,short dbFieldNumber);

int LDADeleteDB(char *DBName);

Delete the database whose name is DBName.

Sample Code:

```
#include <mini_db.h>
static void DeleteDB(void)
{
    int g_db_handle ;
    g_db_handle = LDAOpenDB("address",0);
    if( g_db_handle != -1) // open database sucessfully

    {
        LDACloseDB(g_db_handle,0);
        LDADeleteDB("address");
    }
}
```

int LDACloseDB(DBHANDLE fdidx,short flag);

Close the database.

Sample Code: see `int LDACreateDB(char *DBName,short DBFlags,DBFIELD *dbField,short dbFieldNumber);`

`int LDASetDBCategory(DBHANDLE fdidx,char cate_option);`

It is an important function that will affect the records you want to get next . If you set the category whose ID is cate_option, that means you make this category is current category. And you can only get those records belonging to this category.

Sample Code: see `long LDAGetRecordNumber(DBHANDLE fdidx)`

`int LDASetDBSecurity(DBHANDLE fdidx,char secu_option)`

Security is another special field something like category. But it has only two choices: DB_SHOW_SECURITY or DB_HIDE_SECURITY. If the parameter secu_option is set to DB_HIDE_SECURITY, those records whose property security is DB_RECORD_SECURITY can not be got by those functions such as LDAGetRecord(). And if you use this function and set the secu_option is DB_SHOW_SECURITY, you can get all records.

Sample Code: see `long LDAGetRecordNumber(DBHANDLE fdidx)`

`long LDAGetRecordNumber(DBHANDLE fdidx)`

Get the number of the record whose category is the current category.

Sample Code:

```
#include <mini_db.h>
typedef struct {
    char last_name[25];
    char first_name[25];
}ADDRESS_T; //Correspond to the DBFIELD struct in function 1
static void GetRecordCount(void)
{
    short i;
    int g_db_handle ;
    ADDRESS_T rec1,rec2;
    long num1,num2,total;
    strcpy(rec1.last_name,"Johns");
    strcpy(rec1.first_name,"Smith");
    strcpy(rec2.last_name,"Philips");
    strcpy(rec1.first_name,"Adams")
    g_db_handle = LDAOpenDB("address",0);
    if( g_db_handle != -1) // open database successfully

    {
        cate_index1=LDAAddDBCategory(g_db_handle,"Personal");
        cate_index2=LDAAddDBCategory(g_db_handle,"Business");
```

```
        //normal record
        LDAAddRecord(g_db_handle,&rec1,cate_index1,DB_SHOW_SECURITY));
        //protected record
        LDAAddRecord(g_db_handle,&rec2,cate_index2,DB_RECORD_SECURITY);
        // set the database to protected mode
        LDASetDBSecurity(g_db_handle,DB_HIDE_SECURITY);
        LDASetDBCategory(g_db_handle,cate_index1);
        //There is one record in Personal catalog ,num1=1
        num1=LDAGetRecordNumber(g_db_handle);
        LDASetDBCategory(g_db_handle,cate_index2);
        //There is one record in Business catalog. It is protected, so num2=0
        num2=LDAGetRecordNumber(g_db_handle);
        //total is 2. There are two records.
        total=LDAGetTotalRecords(g_db_handle);
        LDACloseDB(g_db_handle,0);
    }
}
```

long LDAGetTotalRecords(DBHANDLE fdidx)

Get the number of all record in the database which handle is fdidx.

Sample Code: see long LDAGetRecordNumber(DBHANDLE fdidx)

int LDAAddRecord(DBHANDLE fdidx,void *data, char category, char property)

Add a record to the database. **data** is the record pointer and usually it is a structure representing a record. **category** is the id of the category that this record belong to, and the property is the security property which can be DB_SHOW_SECURITY or DB_HIDE_SECURITY. If you set the record property as DB_HIDE_SECURITY, the record is encrypted to some extent, for you can no longer get the record by those functions such as LDAGetRecord() unless you have used the function LDASetDBSecurity();

CAUTION: All the records that we have added are SORTED by their first fields. If we add a new record, it is insert into the database by its first field and other records Ids may be changed.

Sample Code: see long LDAGetRecordNumber(DBHANDLE fdidx)

long LDAFindRecordByValue(DBHANDLE fdidx,void *data,void * CompareValue, char * category, char * property)

Find the record by some value of one field in the database which handle is fdidx. Data is the found record if the function succeed. If failed , return the first record whose field is greater than the **CompareValue** .

CAUTION: now we can only find a record by the first field.

Sample Code:

```
static void SearchRecord(void)
```

```
{
    short search_id,cate_index1,cate_index2;
    ADDRESS_T rec_addr,rec1,rec2;
    char last_name[100],cat,sec;
    strcpy(rec1.last_name,"Johns");
    strcpy(rec1.first_name,"Smith");
    strcpy(rec2.last_name,"Philips");
    strcpy(rec1.first_name,"Adams");
    g_db_handle = LDAOpenDB("address",0);
    if( g_db_handle != -1) //open database successfully

    {
        cate_index1=LDAAddDBCategory(g_db_handle,"Personal");
        LDAAddRecord(g_db_handle,&rec1,cate_index1,DB_SHOW_SECURITY);
        LDAAddRecord(g_db_handle,&rec2,cate_index2,DB_SHOW_SECURITY);
        strcpy(last_name,"Smith");
        //using lastname to find the record , record ID saves at search_id.
        //The record saves at rec_addr
        search_id = LDAFindRecordByValue(g_db_handle,&rec_addr,last_name,&cat,&sec);
        LDACloseDB(g_db_handle);
    }
}
```

int LDAGetNextRecord(DBHANDLE fdidx, void *data, char * category, char * property)

Get the next record . Recommend using this function for it will speed the records searching.

Sample Code: see `int LDAGetRecord(DBHANDLE fdidx,void *data,long RecordId,char * category,char * property)`

int LDAGetRecord(DBHANDLE fdidx,void *data,long RecordId,char * category,char * property)

Get the record by its ID. The function put the found record into the data, category id into **category**, and property into **property**.

SampleCode:

```
static void HandleRecordDemo(void)
{
    short cate_index1,cate_index2;
    ADDRESS_T rec_addr,rec1,rec2;
    char cat,sec;
    strcpy(rec1.last_name,"Johns");
    strcpy(rec1.first_name,"Smith");
    strcpy(rec2.last_name,"Philips");
    strcpy(rec1.first_name,"Adams");
}
```

```
g_db_handle = LDAOpenDB("address",0);
if( g_db_handle != -1) //open database successfully
{
    cate_index1=LDAAddDBCategory(g_db_handle,"Personal");
    LDAAddRecord(g_db_handle,&rec1,cate_index1,DB_SHOW_SECURITY);
    LDAAddRecord(g_db_handle,&rec2,cate_index2,DB_SHOW_SECURITY);
    LDAGetRecord(g_db_handle,&rec__addr,0,&cat,&sec);
    LDAGetNextRecord(g_db_handle,&rec__addr,&cat,&sec);
    LDADeleteRecord(g_db_handle,0);
    LDAUpdateRecord(g_db_handle,&rec1,0,cate_index1,DB_SHOW_SECURITY);
    LDACloseDB(g_db_handle);
}
}
```

int LDADeleteRecord(DBHANDLE fdidx, short RecordId)

Delete a record. And other records Ids may be changed.

Sample Code: see `int LDAGetRecord(DBHANDLE fdidx,void *data,long RecordId,char *category,char * property)`

int LDAUpdateRecord(DBHANDLE fdidx,void *data,short RecordId,char category,char property)

Updated a record. Remember that other record Ids may be changed.

Sample Code: see `int LDAGetRecord(DBHANDLE fdidx,void *data,long RecordId,char *category,char * property)`

8. Other functions

This sections introduces some functions in the other libs you may need. These libs always placed in the directory “opt/src/LinuxDA/pilot/libgui”.

void Showdate(short *year,short *mon, short *mday);

This function will generate a form which shows a calendar to choose a day as Figure 2.1.1 shown . The three parametres will give the date value which you choose in the calendar.

Caution: The data type of Parameters is short ,not int. While the data type of time fields in structure **tm** is int, Remember to convert their types before you using this function

Returns: **0** when you click ‘Cancel/OK’ button to return;

Keycode (KEY_ENTER/KEY_RETURN) when you click especial keys “Enter” and “Return”(on touch screen or hardware key).



Figure 2.1.1

This function is included in a separated lib of “date.o” .If you want to call this function , you **MUST** add this lib to the Makefile file.

In “gui_settime.o”

void gui_settime(time_t *time_set);

This function will generate a form as Figure 2.1.2 shown . You can set time in your application in this form. The parameter “time_set” is time you bring to this functions and it will be displayed when the form is created. After setting time, the time value returned is in “time_set”.

This function return the keycode value if you click “RETURN” key to return.



Figure 2.1.2

9. Standard Linux Lib Functions

This section provides some useful standard linux lib functions.

1.time

1.time

1.1 Fundamental definitions

```
struct tm { // a broken-down time
    int tm_sec; //seconds after the minute. Normally in the range [0 ,59],but can be up to
    61 to allow for leap second.
    int tm_min; //minutes after the hour, [0,59]
    int tm_hour; //hours after midnight.[0,59]
    int tm_mday; //day of the month.[1,31]
    int tm_mon; //month of the year since January. [0,11].
    int tm_year; //the number of year since 1900.
    int tm_wday; //day of week.[0,6].
    int tm_yday; //day in the year. [0,365].
    int tm_isdst; //daylight saving time.
};
```

struct tm representing broken-down time which is binary representation separate into year, month, day etc.Broken-down time is stored in the structure tm which is defined in <time.h>.

time_t a long integer represents calendar time. When interpreted as an absolute time value ,it represents the number of seconds elapsed since 00:00:00 on January 1,1970,Coordinated Universal Time.

2.1 Some functions about time.

struct tm *gmtime(const time_t *timep);

converts the calendar time timep into broken-down time representation, expressed in Coordinated Universal Time.

struct tm *localtime (const time_t *timep);

converts the calendar time timep into broken-down time representation,expressed in local time.

time_t mktime (struct tm *timeptr);

convert a local broken time to a calendar time.

int stime(time_t *t);

set the system time as the value of *t.

time_t time(time_t *t);

Time() returns the time since the Epoch 00:00:00 UTC, January 1, 1970, measured in seconds. If t is non_NULL, the return value is also stored in memory pointed to by t.