WINDOW

Description :

The Window is created on the screen which has a Title Bar and a cross button . The Window serves as a background to the components.

Features :

1. The window created can be hided , freezed ,defreezed and can be deleted .
2. We can also give the window the following :
   - A window caption .
   - A Menu handle .
   - A Child Window .
   - And we can change the color of the screen .

Limitations :

1. Up to 100 windows can be created simultaneously.
2. The Window created should not exceed the maximum height & width of 640 & 480 .
3. Before the parent Window is deleted all the child Window should be deleted .
4. All Windows should be deleted before it terminates otherwise they would remain as a zombie Windows on the screen which would consume vital resources .

WINDOW CREATION :

The following API is used to create the Window :

int Create_Window(unsigned id ,unsigned pid ,unsigned short procid , int x , int y, int w ,int h ,int stat = 1);

The parameters are

- id refers as the identifier value of the Window . It must be unique for every Window created , i.e , the value must not be used by any other Window . It is of the form unsigned integer.
- pid refers to the identifier value of the parent window on which a child Window can be created . It should be '0' for normal Window .
- procid is the value returned by GUIlogin( ) .It returns SS_SUCCESS on success and SS_FAILURE on failure.
- x and y are the left and top screen coordinates where the Window is positioned .
- w and h are the width and the height of the Window .
- stat = 1 is given if the title of Window is required .

Example :

```
Create_Window( WINDOW_1,0,hproc,0,0,640,480,stat = 1);
```

The above API creates a Window at position (0 , 0) with a width of 640 and height of 480 , which is a normal Window with a title .

Displaying The Window :

The following API is used to display the created Window , which is hidden by default ,

void Show_Window(unsigned id , unsigned short procid);

The parameters are :

- Here id refers to the identifier value of the Window which must be displayed .
- procid returns the value SS_SUCCESS if the Window is successfully displayed or else SS_FAILURE is returned if it fails to display the Window .

Example :

```
Show_Window(WINDOW_1,hproc);
```

The above API displays the WINDOW_1 on the screen . And thehproc returns the value as SS_SUCCESS if the Window is displayedor SS_FAILURE if the Window is not displayed .

Hiding The Window :

This API is used to hide the window which is displayed on the screen ,

void Hide_Window(unsigned Id , unsigned short procid);

The parameters in the above API are:

- id is the identifier value of the Window which is to be hided.
- hproc returns the value of the function Hide_Window.

Example :

```
Hide_Window(WINDOW_1,hproc);
```

The above API hides the Window with Identifier Value WINDOW_1.And hproc returns the value of the above function .

NOTE : The hidden window can be retrieved by using the API , SHOW_WINDOW .

Freezing A Window :

The following API is used to freeze the necessary Window.

If a Window is Freezed , only the components in the freezed Window can be accessed and other Window can not be accessed until the Freezed Window is deleted or it is DeFreezed .

void Freeze_Window(unsigned short procid);

The above API does not have any identifier value , it only returns a value for success or failure .

Example:

```
Create_Window(WINDOW_1,0,hproc,0,0,640,480);
Show_Window(WINDOW_1,hproc);
```

Freeze_Window(hproc);

In the above example ,the created Window with identifier value WINDOW_1 is freezed as the API is used after displaying the Window .

No identifier value is required for this API , as the Window will be Freezed , if the API is used after displaying the Window .

<u>DeFreezing The Window :</u>

The following API is used to Defreeze the Window which was freezed before ,

void DeFreeze(unsigned short procid);

The above API only returns the value of the function .

<u>Deleting the Window :</u>

The following API is used to delete the Window,

int Delete_Window(unsigned id , unsigned short procid);

Using above API , a Window can be deleted by specifying the identifier value of the Window which has to be deleted .The API returns SS_SUCCESS or SS_FAILURE .

Example :

Delete_Window(WINDOW_1,hproc);

Here a Window is deleted with a identifier value WINDOW_1.

<u>Setting The Window Title :</u>

The Window Title is set by default as ‘WINDOW’ , to change the Title of the Window the following API is used ,

void Set_Window_Tiltle(unsigned id , unsigned pid , unsigned short procid , char *text);

The parameters are

- id refers to identifier value of the Window , on which the title has to be set.
- pid refrs to the parent identifier value if any or it is given as ‘0’.
- procid is the value returned by GUIlogin( ).
- text is the title of the Window .The title of the Window should not exceed 80 characters .

Example :

Set_Window_Title(WINDOW_1,0,hproc, “EDIT BOX”);

In the above example the Title of the Window is set as “EDIT BOX”.

Navigation :

- Press < ALT + TAB > to view different windows one after other , which were opened simultaneously .

MENU

Description :

The Menu that is available with the GUI server is created on the Window Title Bar. This is like a standard Menu supporting the hotkeys and the various pull down menus which contain the text labels.

The menu described here is static , i.e , the Text Labels described in the pull down Menu can not be changed dynamically.

Features :

1. A Menu Item or a Menu Page on the menu can be activated or deactivated .
2. The number of items in a page comprises of menu-items , menu-headers , and separators .
3. It has a pull down menu with the text labels which is supported by the hotkeys .

Creating A Menu :

The following API is used to create the Menu , by using the file name which contains the description of the Menu which is given in the API .

short Create_Menu(unsigned id , unsigned pid , unsigned hproc , int x , int y , int w , int h , char *filename);

The parameters are :
id refers to the identifier value of the Menu
pid refers to the identifier value of the parent Window on which the Menu has to be created .

- hproc returns the value of the API .
- x & y refers to the coordinates of the Menu on the upper left corner of the screen.
- w & h are the height and width of the Menu .
- filename refers to the file which has the description of the Menu.

Format of the Menu descriptor file:

At the beginning of the file it should always start with MENU : followed by the total number of menu pages to be created.

Format of the menu page

```
MENU :    [1] ;
"<name  of  page >"[ <number of  items  in  the page>][<identifier value>]{
                "name   of  menu  item>"[<identifier  value>];
                "name   of  menu  item>"[<identifier  value>];
                "name   of  menu  item>"[<identifier  value>];  }
```

- The < identifier value > is an integer value and is returned by the menu to the application process when the corresponding menu item is selected by the user .
- The identifier value can also be given to the menu page ,if necessary .
- The value < number of items in the page > denotes the total number of items in the page .
- Separators if any are defined by keyboard (/SEPARATORS) and these are also taken in to account when stating the number of items in the page .
- Every menu page must be enclosed in the curly braces , i.e , '{ } ' . And no semicolon must be inserted after the curly brace .
- Each menu item with the identifier value must have an semicolon at the end .
- Each label must be enclosed in the double quotes containing at least one character .
- By default the character of each label is taken as the hotkey . This can be overridden by entering '&' before the desired character in the label .

Example :

The example for how to create menu is given below ,

```
MENU : [5] ;
```

```
“Main” [2] {
        “File” [4][1001] {
                    “New”[101];
                    “Open”[102];
                    /SEPARATOR
                    “Exit”[103];   }
        “Edit”[4]{
                   “Cut”[201];
                   “Co&py”[202];
                   “Paste” [203];
                    “Send  &To”[2]{
                                 “Desktop”[301];
                                 “My Documents”[302];}
                  }
           }
```

In the above example a menu is created with two menu pages “File” & “Edit” , and these pages contains four items each .And a sub-menu is also created in the menu page “Edit” as “Send to” which has two items .

<u>To Activate & Deactivate An Item :</u>

Using the following API an item can be Activated or deactivated , if the item is deactivated , it means , the item can not be accessed and it can also be activated .

void Set_Menu_Item(unsigned id , unsigned pid , unsigned short procid , unsigned value , short state);

In the above API :

- id refers to the identifier value of the menu .
- pid refers to the identifier value of the parent window on which the menu is created .
- procid is the value returned by GUIlogin( ).
- value is the identifier value of the menu item, which is to be activated or deactivated .
- If state = 1 , then the menu item is activated or the menu item is deactivated , i.e , if state = 0 .

Example :

Set_Menu_Item(MENU_1,WINDOW_1,hpro,101,0);

In the above example , suppose the menu in the example is MENU_1 in window WINDOW_1 . Then using identifier value 101 , the item "New" in item page "File" can be deactivated .

To Activate or Deactivate a menu page :

Using the following API a menu page can be activated or deactivated ,

void Set_Menu_Page(unsigned id , unsigned pid , unsigned short procid , unsigned value , short state);

- id refers to the identifier value of the menu .
- pid refers to the identifier value of the parent window on which the menu is created .
- procid is the value returned by GUIlogin( ).
- value is the identifier value of the menu page which is to be activated or deactivated .
- If state = 1 , then the menu page is activated or the menu item is deactivated , i.e , if state = 0 .

Example :

Set_Menu_Item(MENU_1,WINDOW_1,hproc,1001,0);

The menu page with the identifier value 1001 is deactivated.

Navigation :

The Menu contents can be navigated using the arrow keys. Using the right and left arrow keys, one can shift the focus to other menu pages . The highlighted bar shows that the
particular item or page is in focus . To view the items in the menu page , get the menu page in to focus and just press <Enter> or press down arrow key .

CANVAS

Description :

The canvas is a component which is created on the screen of the window as an image such that one can draw line , circle ,text ,bar , pixel on the canvas .

Creating a Canvas :

The following API is used to create the canvas on the window of the screen ,

int Create_Canvas(unsigned id , unsigned pid , unsigned short procid, int x , int y , int w , int h);

Where the parameters are ,

- id refers to the identifier value of the canvas .
- pid refers to the identifier value of the parent window on which the canvas is created .
- procid is the value returned by the GUIlogin( ). It returns the value as SS_SUCCESS for success or else it returns SS_FAILURE .
- x & y are the top left corner coordinates of the screen .
- w & h are the width and height of the canvas .

Example :

Create_Canvas(CANVAS_1,WINDOW_1,hproc,0,15,640,480);

Using the above example a canvas CANVAS_1 can be created on the screen of the window WINDOW_1 .

API's used to Draw in the canvas :

Line :

To draw a line from a point or coordinate (x1 , y1) to the point (x2 , y2) on the canvas , the following API is used ,

int Draw_Line(unsigned id , unsigned pid , unsigned short procid ,int x1 , int y1 , int x2 ,int y2 ,int color);

The parameters used are,

- id refers to the identifier value of the canvas on which the line is drawn .

- pid refers to the identifier value of the parent window on which the canvas is created .
- procid is the value returned by the GUIlogin( ).
- x1 & y1 are the starting points of the line and x2 & y2 are the ending points till were the the line is drawn .
- color is used to set color of the line .

Example :

```
Draw_Line(CANVAS_1,WINDOW_1,hproc,0,0,200,250,1);
```

In the above example the line is drawn starting from the point (0,0) till the point (200,250) , and the line drawn on the canvas CANVAS_1 is blue in color .

Circle :

The following API is used to draw a circle on the canvas with the radius given ,

int Draw_Circle(unsigned id , unsigned pid , unsigned short procid , int x , int y , int radius , int color);

The parameters here are ,

- id refers to the identifier value of the canvas on which the circle is drawn .
- pid refers to the identifier value of the parent window on which the canvas is created .
- procid is the value returned by the GUIlogin( ).
- x & y are the center points of the circle .
- Using the given radius , a circle is drawn at the point x & y .

Note : If the radius of the circle exceeds the maximum resolution then the circle is clipped off to certain extent .

Example :

```
int Draw_Circle( CANVAS_1,WINDOW_1,hproc,100,100,20,0);
```

In the above example , the circle with radius 20 is drawn on the canvas CANVAS_1 at a point (100,100) , which is set to black in color (black=0 , as in above API).

Bar :

The following API is used to draw a bar on the canvas ,

int Draw_Bar(unsigned id , unsigned pid , unsigned short procid ,int x ,int y , int w ,int h ,int color);

The parameters here are ,

- id refers to the identifier value of the canvas on which the bar is drawn .
- pid refers to the identifier value of the parent window on which the canvas is created .
- procid is the value returned by the GUIlogin( ).
- x & y are the points at which the bar has to be created .
- w & h are the height and width of the bar .
- It can be set to the desired color by giving the number to the color in the API .

Example :

```
Draw_Bar( CANVAS_1,WINDOW_1,hproc,0,15,640,480,15);
```

In the above example the bar is created at a point (0,15) with width and height of 640 & 480 , which is drawn in white .

Text :

The following API is used to draw a text on the canvas which allows a maximum of 80 characters with a selected font ,

int Draw_Text(unsigned id , unsigned pid , unsigned short procid , int x , int y , char *text);

The parameters here are ,

- id refers to the identifier value of the canvas on which the text is drawn .
- pid refers to the identifier value of the parent window on which the canvas is created .
- procid is the value returned by the GUIlogin( ).
- x & y are the points where the text is drawn .
- text is a string that is to be displayed . Which must be less than 80 characters . The text given must be enclosed in double quotes .

Example:

```
Draw_Text( CANVAS_1,WINDOW_1,hproc,100,100,” New  Window”);
```

In the above example a text is drawn on the canvas at a point (100,100) .

Pixels :

A point or number of points can be drawn on the canvas using the following API ,

int Draw_Pixel(unsigned id , unsigned pid , unsigned short procid , int x , int y , int color);___

The parameters here are ,

- id refers to the identifier value of the canvas on which the pixels are drawn .
- pid refers to the identifier value of the parent window in which the canvas is created .
- procid is the value returned by the GUIlogin( ).
- x & y are the points where the pixels are drawn .
- Pixel can also drawn in desired color .

Example:

```
Draw_Pixel( CANVAS_1,WINDOW_1,hproc,120,20,1);
```

A pixel is drawn at a point (120,20) with a blue color .

Setting the Text Color :

The fore ground , back ground and font of the text on the canvas can be set using the following API ,

int Set_Text_Colors(unsigned id , unsigned pid , unsigned short procid , unsigned fc , unsigned bc , unsigned fno);

where

- id refers to the identifier value of the canvas.
- pid refers to the identifier value of the parent window .
- procid is the value returned by the GUIlogin( ).
- fc refers to the fore ground color of the text on the canvas .
- bc refers to the back ground color of the Text on the canvas .
- fno refers to the font number to which the text is set .

The fc , bc and fno set to a canvas remains the same for any number of texts drawn on the canvas .

Example:

```
while(1)
{
        Set_Text_Colors(CANVAS_1,WINDOW_1,procid,count % 255,
backcolor , rand ( ) %  MAX_FONT_NO);
 Draw_Text(CANVAS_1,WINDOW_1,hproc,100,100, “HELLO”);
 }
```

The above code draws a text “HELLO” in random colors and in different fonts at position (100,100) in the canvas whose id is CANVAS_1 in a window whose id is WINDOW_1.

The “MAX_FONT_NO “ which is used in the above code is 30 . The number of fonts available are 30 (0 to 29).

The FONTS size and type are given below :

Font No.	Font Type	Font Size
0	“Times Roman”	10
1	“Times Roman”	12
2	“Times Roman”	14
3	“Times Roman”	16
4	“Times Roman”	20
5	“Times Roman”	24
6	*“Times Italic”*	10
7	*“Times Italic”*	12
8	*“Times Italic”*	14
9	*“Times Italic”*	16
10	“Times italic”	20
11	*“Times Italic”*	24
12	“Courier”	10
13	“Courier”	12
14	“Courier”	16

Font No .	Font Name	Font Size
15	“Courier”	20
16	“Courier”	24
17	*“Arial Italic”*	10
18	*“Arial Italic”*	14

19	*“Arial Italic”*	16
20	*“Arial Italic”*	24
21	“Impact”	12
22	“Impact”	16
23	“Impact”	20
24	“Impact”	24
25	“Arial”	12
26	“Arial”	14
27	“Arial”`	16
28	“Arial”	20
29	“Arial”	24

Using any font number from 0 – 29 ,the desired text can be drawn in the canvas . The total number of colors available are
256(0 – 255).

<u>API’s Used to Shift the Contents :</u>

The following API’s are used to shift the components of a canvas by the desired amount to the left , right ,top & bottom.

<u>UP :</u>

The following API is used to shift the contents to the top of the canvas by a desired amount with filling the cleared portion with color ,

void Shift_Up_By(unsigned id, unsigned pid ,unsigned short procid ,unsigned dy , unsigned color);

where

- id refers to the identifier value of the canvas.

- pid refers to the identifier value of the parent window .
- procid is the value returned by the GUIlogin( ).
- dy refers to the amount by which the canvas contents are to be shifted vertically.
- color is used to fill the cleared portion with the desired color value.

Example :

```
Shift_Up_By(CNAVAS_1,WINDOW_1,hproc,100,15);
```

It shifts the contents of the canvas up by 100 filling the cleared portion with white color .

DOWN :

The following API is used to shift the contents to the bottom of the canvas to the desired amount .

void Shift_Down_By(unsigned id , unsigned pid , unsigned short procid , unsigned dy , unsigned color);
where

- id refers to the identifier value of the canvas.
- pid refers to the identifier value of the parent window .
- procid is the value returned by the GUIlogin( ).
- dy refers to the amount by which the canvas contents are to be shifted vertically.
- color is used to fill the cleared portion with the desired color value.

Example :

```
Shift_Down_By(CANVAS_1,WINDOW_1,hproc,100,15);
```

It is used to shift the contents to the bottom of the canvas by 100 and filling the cleared portion with white color .

RIGHT:

The following API is used to shift the contents to the right of the canvas by a desired amount with filling the cleared portion with color ,

void Shift_Right_By(unsigned id, unsigned pid ,unsigned short procid ,unsigned dx , unsigned color);

where

- id refers to the identifier value of the canvas.
- pid refers to the identifier value of the parent window .
- procid is the value returned by the GUIlogin( ).
- dx refers to the amount by which the canvas contents are to be shifted horizontally.
- color is used to fill the cleared portion with the desired color value.

Example :

```
Shift_Right_By(CNAVAS_1,WINDOW_1,hproc,100,15);
```

It shifts the contents to the right of the canvas by 100 filling the cleared portion with white color .

LEFT :

The following API is used to shift the contents to the left of the canvas by a desired amount with filling the cleared portion with color ,

void Shift_Left_By(unsigned id, unsigned pid ,unsigned short procid ,unsigned dx , unsigned color);

where

- id refers to the identifier value of the canvas.
- pid refers to the identifier value of the parent window .
- procid is the value returned by the GUIlogin( ).
- dx refers to the amount by which the canvas contents are to be shifted horizontally.
- color is used to fill the cleared portion with the desired color value.

Example :

```
Shift_Left_By(CNAVAS_1,WINDOW_1,hproc,100,15);
```

It shifts the contents to the left of the canvas by 100 filling the cleared portion with white color .

<u>Setting Key Board messages ON / OFF :</u>

The following API is used to enable/disable the keyboard from the server when the focus is on the canvas ,

void Set_Canvas_Key_State(unsigned id, unsigned pid , unsigned short procid , int keyState);

where

- id refers to the identifier value of the canvas.
- pid refers to the identifier value of the parent window .
- procid is the value returned by the GUIlogin( ).
- If keyState = 1 , the key flow is enabled or it is disabled . By default the keyboard messages is set off.

Example:

```
Set_Cnavas_key_State(CANVAS_1,WINDOW_1,hproc,1);
```

The keyboard message state is set enabled in the canvas with id CANVAS_1 in the window with id WINDOW_1.

<u>SCROLLABLE CANVAS</u>

Description :

The scrollable canvas is much like a canvas . It defines a logical display area which is larger than the one which is available. It has the same features as that of canvas.

Creation:

The following API is used to create a scrollable canvas ,

int Create_ScrollableCanvas(unsigned id , unsigned pid , unsigned short procid , int x , int y , int w ,int h , int vw , int vh);

where the parameters are:

- id refers to the identifier value of the canvas .
- pid refers to the identifier value of the parent window on which the scrollable canvas is created .
- procid is the value returned by the GUIlogin( ). It returns the value as SS_SUCCESS for success or else it returns SS_FAILURE .
- x & y are the top left corner coordinates of the screen .
- w & h are the width and height of the scrollable canvas .
- vw , vh are the virtual width and virtual height of the scrollable canvas .

Example:

Create_ScrallableCanvas(CANVAS_1,WINDOW_1,hproc,20,30,640,480,100,100);

Above example is used to create a scrollable canvas at a position (20,30) with a height and width of (640,480).And the vertical width and height of the canvas is given as (100,100).

API's used to draw in the Scrollable canvas:

Line :

To draw a line from a point or coordinate (x1 , y1) to the point (x2 , y2) on the scrollable canvas , the following

API is used ,

int Draw_LineSc(unsigned id , unsigned pid , unsigned short procid ,int x1 , int y1 , int x2 ,int y2 ,int color);

The parameters used are,

- id refers to the identifier value of the scrollable canvas on which the line is drawn
- pid refers to the identifier value of the parent window on which the scrollable canvas is reated .
- procid is the value returned by the GUIlogin( ).
- x1 & y1 are the starting points of the line and x2 & y2 are the ending points till were the the line is drawn .
- color is used to set color of the line .

Example :

Draw_Line(CANVAS_1,WINDOW_1,hproc,0,0,200,250,1);

In the above example the line is drawn starting from the point (0,0) till the point (200,250) , and the line drawn in the canvas CANVAS_1 is blue in color .

<u>**Circle :**</u>

The following API is used to draw a circle on the canvas with the radius given ,

int Draw_CircleSc(unsigned id , unsigned pid , unsigned short procid , int x , int y , int radius , int color);

The parameters here are ,

- id refers to the identifier value of the canvas on which the circle is drawn .
- pid refers to the identifier value of the parent window on which the canvas is created .
- procid is the value returned by the GUIlogin( ).
- x & y are the center points of the circle .
- Using the given radius , a circle is drawn at the point x & y .

Example :

int Draw_CircleSc(CANVAS_1,WINDOW_1,hproc,100,100,20,0);

In the above example , the circle with radius 20 is drawn on the canvas CANVAS_1 at a point (100,100) , which is set to black in color (black=0 , as in above API).

<u>**Bar :**</u>

The following API is used to draw a bar on the scrollable canvas ,

int Draw_BarSc(unsigned id , unsigned pid , unsigned short procid ,int x ,int y , int w ,int h ,int color);

The parameters here are ,

- id refers to the identifier value of the scrollable canvas on which the bar is drawn .
- pid refers to the identifier value of the parent window in which the scrollable canvas is created .
- procid is the value returned by the GUIlogin( ).
- x & y are the points at which the bar has to be created .
- w & h are the height and width of the bar .
- It can be set to the desired color by giving the number to the color in the API .

Example :

Draw_BarSc(CANVAS_1,WINDOW_1,hproc,0,15,640,480,15);

In the above example the bar is created at a point (0,15) with width and height of 640 & 480 , which is drawn in white .

<u>Text :</u>

The following API is used to draw a text on the scrollable canvas which allows a maximum of 80 characters with a selected font , **int Draw_TextSc(unsigned id , unsigned pid , unsigned short procid , int x , int y , char *text);**

The parameters here are ,

- id refers to the identifier value of the scrollable canvas on which the text is drawn
- pid refers to the identifier value of the parent window on which the scrollable canvas is created .
- procid is the value returned by the GUIlogin( ).
- x & y are the points where the text is drawn .
- text is a string that is to be displayed . Which must be less than 80 characters . The text given must be enclosed in double quotes .

Example:

Draw_TextSc(CANVAS_1,WINDOW_1,hproc,100,100,” New Window”);

In the above example a text is drawn on the scrollable canvas at a point (100,100) .

<u>Pixels :</u>

A point or number of points can be drawn on the scrollable canvas using the following API ,

int Draw_PixelSc(unsigned id , unsigned pid , unsigned short procid , int x , int y , int color);___

The parameters here are ,

- id refers to the identifier value of the scrollable canvas on which the pixels are drawn .

- pid refers to the identifier value of the parent window in which the scrollable canvas is created .
- procid is the value returned by the GUIlogin( ).
- x & y are the points where the pixels are drawn .
- Pixel can also drawn in desired color .

Example:

Draw_PixelSc(CANVAS_1,WINDOW_1,hproc,120,20,1);

A pixel is drawn at a point (120,20) with a blue color .

Setting the Text Color :

The fore ground , back ground and font of the text in the scrollable canvas can be set using the following API ,

int Set_Text_Colors(unsigned id , unsigned pid , unsigned short procid , unsigned fc , unsigned bc , unsigned fno);

where

- id refers to the identifier value of the scrollable canvas.
- pid refers to the identifier value of the parent window .
- procid is the value returned by the GUIlogin( ).
- fc refers to the fore ground color of the text in the scrollable canvas .
- bc refers to the back ground color of the Text in the scrollable canvas .
- fno refers to the font number to which the text is set .

The fc , bc and fno set to a scrollable canvas remains the same for any number of texts drawn in the scrollable canvas .

Setting Key Board messages ON / OFF :

The following API is used to enable/disable the keyboard from the server when the focus is on the scrollable canvas ,

void Set_Canvas_Key_State(unsigned id, unsigned pid , unsigned short procid , int keyState);

where

- id refers to the identifier value of the scrollable canvas.

- pid refers to the identifier value of the parent window .

- procid is the value returned by the GUIlogin( ).
- If keyState = 1 , the key flow is enabled or it is disabled . By default the keyboard messages is set off.

Example:

Set_Cnavas_key_State(CANVAS_1,WINDOW_1,hproc,1);

The keyboard message state is set enabled in the scrollable canvas with id CANVAS_1 in the window with id WINDOW_1.

Navigation :

- On selecting the scrollable the arrow keys may be used to move around the scrollable canvas .

LABLE

Description :

A label is used to display a text any where in the window.

Creating Label :

The following API is used to draw a text on the window ,

Create_Label(unsigned id, unsigned pid , unsigned short procid, int x, int y, int w, int h, char *text);

Where the parameters are ,

- id refers to the identifier value of the label .

- pid refers to the identifier value of the parent window , in which the label is created .

- procid is the value returned by the GUIlogin( ).

- x , y refers to the position of the label in the window.

- h and w refers to the height and width of the label.

- text refers to the string that is to be displayed in the window . Text supports a string of 80 characters only.

Example :

Create_Label(LABEl_1,WINDOW_1,hproc,100,100,100,10,"HELLO");

A label "HELLO" with identifier value LABEL_1 can be created in the window whose id is WINDOW_1 at a position (100,100) with width of 100 and height of 10 , using the above api .

Changing the text :

This API is used to change the text ,

-

int Set_Text_Label(unsigned id , unsigned pid , unsigned short procid ,char *text);

Where

- id refers to the identifier value of the label .

- pid refers to the identifier value of the parent window , in which the label is changed.

- procid is the value returned by the GUIlogin( ).

- text is the string of characters which are used to change the previous text.

Example :

```
Set_Text_Label(LABEL_2,WINDOW_1,hproc,”WORLD”);
```

Using the above API we can change the text we have created in the window whose id is WINDOW_1 .

NOTE: 1. The text must be in always double quotes or it supports character buffer[ ].
2. The identifier value must be unique for each label.

BUTTON

Description :

Buttons are windows components whose responses are used to trigger events when they are selected and < ENTER > is pressed .

Creation :

The following API is used to create a button in the window,

short Create_Button(unsigned id , unsigned pid , unsigned short procid , int x , int y , int w , int h , char *text);

The parameters are :

- id refers to the identifier value of the button . This identifier value must be unique and must be defined at the starting of the program .
- pid refers to the identifier value of the parent window ,

in which the button is created .

- procid is the value returned by the GUIlogin( ).
- x , y refers to the starting position of the button in the window.
- h and w refers to the height and width of the button.
- text refers to the string that is to be displayed in the button . Text supports a string of 80 characters only.

Example:

Create_Button(BUTTON_1,WINDOW_1,hproc,100,100,30,100,"Ok");

The above example creates a button in the window whose id is WINDOW_1 at the position (100,100) with height 30 and a width of 100,
whose label is set as "Ok".

Setting the Button text :

The following API is used to change the label of the button whose label is set before,

int Set_text_Button(unsigned id , unsigned pid , unsigned short procid ,char *text);

The parameters are :

- id refers to the identifier value of the button . This identifier value must be unique and must be defined at the starting of the program .
- pid refers to the identifier value of the parent window , in which the button is created .
- procid is the value returned by the GUIlogin( ).
- text refers to the string that is to be displayed in the button . Text supports a string of 80 characters only.

EDIT BOX

Description :

The following API is used to edit a string of characters which is taken as a user text information. API's are used to set the text in the edit box and can even retrieve the text from the edit box.

Creation:

The following API is used to create a edit box ,

int Create_EditBox(unsigned id , unsigned pid , unsigned short procid , int x , int y ,int w ,int h);

where the parameters are :

- id refers to the identifier value of the edit box.
- pid refers to the identifier value of the parent window , in which the edit box is created .
- procid is the value returned by the GUIlogin( ).
- x , y refers to the position of the edit box in the window.
- h and w refers to the height and width of the edit box .

Example:

```
Create_EditBox(EDITBOX_1,WINDOW_1,hproc,100,100,80,30);
```

The above example is used to create edit box at a position (100,100) with width of 80 and with a height of 30 in the window whose identifier value is WINDOW_1.

Setting the text in Edit Box :

The text can be displayed in the edit box using the following API ,

int Set_Text_EditBox(unsigned id , unsigned pid , unsigned short procid , char *text);

where the parameters are :

- id refers to the identifier value of the edit box.
- pid refers to the identifier value of the parent window , in which the edit box is created .
- procid is the value returned by the GUIlogin( ).
- The text here is a buffer of characters from which the text is set .

Example:

```
 char   Buffer[20];
sprintf( Buffer," %d  ", " SET TEXT");
Set_Text_EditBox(EDITBOX_1,WINDOW_1,hproc,Buffer);
```

This code sets a text "SET TEXT" in the edit box .The text is copied in to a Buffer which is declared as a character in the starting of the code and the Buffer is used to set the text.

Retrieving the Edit Box Text :

The following API is used to retrieve the text from the edit box,

int Get_Text(unsigned id , unsigned pid , unsigned short procid ,

char *Buffer);

where the parameters are :

- id refers to the identifier value of the edit box.
- pid refers to the identifier value of the parent window , in which the edit box is created .
- procid is the value returned by the GUIlogin( ).
- The text here is a buffer of characters from which the text is retrieved .

<u>Example:</u>

```
  char  Buffer1[20];
Get_Text(EDITBOX_1,WINDOW_1,hproc,Buffer1);
```

This code gets the text from the edit box and is kept in the Buffer1 which is declared as an character with the array size of 20.

NOTE: Up to 256 characters can be kept in the text or buffer .

PASSWORD BOX

Description :

PassWord Box is much like a Edit box but the text entered in to the password box is replaced with the asterisks .The text entered in to box is taken as user password.

Creation:

The following API is used to create a password box ,

int Create_Passwordbox(unsigned id, unsigned pid , unsigned short procid , int x , int y , int w , int h);

where the parameters are :

- id refers to the identifier value of the password box .
- pid refers to the identifier value of the parent window , in which the password box is created .
- procid is the value returned by the GUIlogin( ).
- x , y refers to the position of the password box in the window.
- h and w refers to the height and width of the

password box .

Example:

```
Create_PasswordBox(EDITBOX_2,WINDOW_2,hproc,120,200,100,40);
```

The above API is used to create a password box at position (120,120) with height 40 and with width of 100.

Setting Text :

To set the password box to an initial value , the following API is used ,

int Set_Text_passwordBox(unsigned id , unsigned pid , unsigned short procid , char *text);

where the parameters are :

- id refers to the identifier value of the edit box.
- pid refers to the identifier value of the parent window , in which the edit box is created .
- procid is the value returned by the GUIlogin( ).
- The text here is a buffer of characters from which the text is set .

Example:

```
 char Buffer[20];
sprintf( Buffer,” %d “, “CHECK OUT”);
Set_Text_PasswordBox(EDITBOX_1,WINDOW_1,hproc,Buffer);
```

This code sets a text “CHECH OUT” in the edit box ,but each character is replaced with an asterisk in the text. The text is copied in to a Buffer which is declared as a character in the starting of the code and the Buffer is used to set the text.

Retrieving the Password Box Text :

The following API is used to retrieve the text from the edit box,

int Get_Text(unsigned id , unsigned pid , unsigned short procid , char *Buffer);

where the parameters are :

- id refers to the identifier value of the password box.
- pid refers to the identifier value of the parent window , in which the password box is created .
- procid is the value returned by the GUIlogin( ).
- The text here is a buffer of characters from which the text is retrieved .

Example:

```
  char  Buffer1[20];
Get_Text(EDITBOX_1,WINDOW_1,hproc,Buffer1);
```

This code gets the text from the password box and is kept in the Buffer1 which is declared as an character with the array size of 20.

NOTE: Up to 256 characters can be kept in the text or buffer .

CHECK BOX

Description :

A check box is a square box with a check and uncheck mark in the box accomplished with a text label .

Creation :
For the check boxes the following API are used ,

short Create_Chekbox(unsigned id , unsigned pid, unsigned short procid , int x ,int y , int w, int h , char *text);

where the parameters are :

- id refers to the identifier value of the Check box .
- pid refers to the identifier value of the parent window , in which the check box is created .
- procid is the value returned by the GUIlogin( ).
- x , y refers to the position of the check box in the window.
- h and w refers to the height and width of the check box .
- text refers to the label which should not be more than 80 characters associated with the check box .

Example:

Create_CheckBox(CHECKBOX_1,WINDOW_1,hproc,250,100,140,30);

The above API is used to create a check box in window whose id is WINDOW_1 . Iy=t is created at a position (250,100) with a height and width of 30 and 140.

Getting The Check Box Status :

The following API is used to set the status of check box ,

short Get_CheckBox_Status(unsigned id ,unsigned pid ,unsigned short procid , short *pstatus);

where the parameters are :

- id refers to the identifier value of the Check box .
- pid refers to the identifier value of the parent window , in which the check box is created .
- procid is the value returned by the GUIlogin( ).
- pstatus gets the status of the checkbox.

Example:

Get_Check_Status(CHECKBOX_1,WINDOW_1,hproc,status);

which gets the status of the check box.

Seting the Check Box Status :

Set_CheckBox(unsigned id , unsigned pid , unsigned short procid , short status);

This API is used to set the status of the check box. Here the parameters are ,

- id refers to the identifier value of the Check box .

- pid refers to the identifier value of the parent window, in which the check box is created.
- procid is the value returned by the GUIlogin( ).
- status refers to the state 1 for checked and 0 for unchecked states.

<u>Example:</u>

```
Set_CheckBox(CHECKBOX_1,WINDOW_1,hproc,1);
```

In the above API the check box status is set to ‘1’, that means, the check box status is checked.

<u>PROCESS DIAGRAM</u>

<u>GUI server</u>

Application Process

Key Board Process (mq1)

REQUEST QUEUE

OUTPUT QUEUE

GUI SERVER
(SSGUI)

An overview of the internal architecture of the GUI Server

Message Passing:

The GUI or Display Server is an independent process that runs in the background and waits for inputs from its clients or from the foreground input (keyboard) process.

Hence the system can be viewed in the following way.

1. **<u>The display server :</u>**

-

This is a 'window'-ing program that has control over the monitor and which responds to various 'screen' related events < lines, texts, window creation etc > . This program responds appropriately to the user's input too and forwards them to proper client processes for whom these inputs are meant. System related messages are also sent to the appropriate client in response to events < e.g. screen refresh or redraw etc > .

The display server communicates with the outside world via 2 queues.

i. **<u>REQUEST QUEUE:</u>** This queue is where the clients place their requests to the display server.

ii. **<u>OUTPUT QUEUE:</u>** The application processes or client receive feedbacks (in response to certain requests) or input notification, or any other event triggered by the user < e.g. clicking a button, or switching the active window > from this queue.

<u>b) The foreground keyboard process :</u>

This process has control over the keyboard and places the user's keystrokes in the REQUEST QUEUE for the display server to pick up .It has a unique message type to distinguish it from other requests

<u>c) Application or Client processes:</u>

These processes logon to the display server and utilize the capacities of the display server in a manner fit for the application. Requests are sent via the REQUEST QUEUE while output from the server comes from the OUTPUT QUEUE. The GUI Server signs each output by the processes id (hproc value returned during GUILogIn)of the client or by a process-id – request-no combination. The client processes poll on the output queue to see if there is any message for it.

Thus the three functional components run independently of one another ... only communicating via messages. Thus we have the following process diagram .

DATA FLOW DIAGRAM

computer screen

GUI SERVER

Output Queue
Request Queue

Client 1
Client n
Client 2

From
Keyboard

Drawing Internals :

Objects:

The different visual components on the screen are derived from the base class *Win_Object* . This include labels, canvases, buttons,checkboxes and menus. Also each window is derived from the base class *Win_Object.*

<u>Exposed Areas</u>

The portion of the screen occupied by each window is maintained in the global object *SS_WinArea* (an instance of the winarea class.)

The *winarea* class maintains a list of lines which represents the boundaries occupied by each window. Hence whenever any drawing activity is to be performed on a window, it's boundaries are looked up from *SS_WinArea* , and the activity is performed in the visible portions of the window and blocked otherwise.

<u>Maintaining the windows :</u>

The different windows are maintained in a list by the *SuperStructure* class . Each node is of type *WinList* which has a pointer to a window and a list containing the various components of the window.

The *<u>SuperStructure</u>* class implements routines to insert ,delete ,hide and show various windows and and its components. Also proper updation of the screen and coordinating with *SS_WinArea* (to ensure proper overlapping of the windows and window ordering)

The interaction between the various classes could be shown as follows

<u>Interaction between the various components of the window manager system</u>

SCREEN

WINAREA

Draw in Appropriate Regions on the screen

Get appropriate areas
for drawing on the window

Search for the window

A REQUEST
FOR DRAWING

The Window List (sorted on procid)

Request handling :

Requests from clients are done by a set of routines

Window System :

"Run_Window_System" routine implements the main loop of the GUI Server. It scans for requests from the keyboard/input queue (using *GetInput*) and the request queue (using *GetSystemInput*).

On receiving a key-stroke it first checks if the keystroke is for the server

e.g. *ALT+ TAB*	: for window switching
or *ALT + 0*	: for GUI shutdown
or *ALT+M*	: for Window move start

and works accordingly. Otherwise the keystoke is sent to *ProcessKeyboard* for further processing on the currently focused window e.g. button clicks etc.

On the other hand if a Request from Client is found then it is passed to the *ServeGUIRequest* function for handling.

Keyboard Process :

"ProcessKeyboard" routine is responsible for sending the accepted key-stroke to the currently focused component of the currently active window. It is also responsible for switching among the currently focused component of a window when a TAB key is hit.

Serving the GUI Request :

"ServeGUIRequest" routine is responsible for the finding out the type of request that the client(s) have sent , and executing the valid ones.

All the above routines communicate with the windows and components using methods of the Super Structure class.

Optimizations :

Following version 2.7 the following optimizations have been made.

i. *Search for a canvas in response to a drawing requests has been implemented using the standard C++ "map" implementation. Thereby reducing response time*
ii. *The server now uses 2 queues instead of 3.*
iii. *The server now block waits on the REQUEST QUEUE (not polling) so that vital CPU resources are not used up unnecessarily.*

TABLE EDITOR

Description :

Table Editor is a built - in Editor which has rows and columns , in which , the data entered is saved . In this the rows can be set to enter the data as numeric or alpha numeric .

Features :

- The number of rows and columns which are created , can be set to the desired value.
- Each column can be set to enter the data as numeric (only numbers) or alpha numeric(Alphabets and numbers).
- Each column can be set as editable or non editable.
- The data entered in to the editor is saved in a data file and it is retrieved when ever the editor is opened.

Creation :

The following API is used to create table editor,

Create_TableEditor(unsigned id , unsigned pid , unsigned short procid, int x , int y ,char *rcfilename;

Where the parameters are

- id refers to the identifier value of the table editor .
- pid refers to the identifier value of the parent window on which the table editor is created.
- procid is the value returned by GUIlogin( ).
- x , y denotes the starting screen coordinates of the table editor in the parent window.
- This is the resource file mane which contains the necessary information regarding the table editor structure , i.e , regarding the number of columns

, rows and there specifications.

Saving the Table Editor's information :

To save the table editor data in a data file the following API should be used ,

```
short Save_Table_Editor(unsigned id , unsigned pid , unsigned short procid , char *datafilename);
```

Where the parameters are

- id refers to the identifier value of the table editor .
- pid refers to the identifier value of the parent window on which the table editor is created.
- procid is the value returned by GUIlogin( ).
- "datafilename" is the string containing the name of the data file in which table editor data would be stored .

Rules for writing the table editor script :

To write a table editor script the following rules should be followed ,

1. The very first line of the table editor script should contain a signature TABLEEDITOR only .

NOTE : The script is case sensitive .

2. On the second line the number of rows and columns of the table editor should be specified. The number should be separated by a blank.

3. Then follows ,the column description which has three fields

    a) Its type , whether the data entered is numeric or nonnumeric. Its denoted as,

        A - alphanumeric
        N - numeric

    b) The length of the column is given .

c) The third field is used to lock the column . If the field is locked then it cannot edit in the column otherwise it is editable .

Y - not editable.

N - editable.

d) Following the column description the option load filenames should be specified as follows ,

```
LOADFILE=loadfilename  with  full path.
```

If no load file is specified the editor will be filled with blank entries. And the loadfilename should be specified with full path name that too with out any blank spaces in between .

<u>Example :</u>

```
TALEEDITOR
10 4
 N 10 Y
 N 10 N
 A 5 N
 A 8 N
  LOADFILE=/gui2.8/demo/table.dat
```

Here is a sample table editor script , the above table editor has 10 rows and 4 columns . First two columns are numeric type of length 10 each and the rest are alphanumenric type of length 5 and 8. The first column is not editable and the last three columns are editable .

The load file here is table.dat which is specified with the full path name /gui2.8/demo.

<u>Limitations :</u>

1. The data file from which the table editor reads the data should be a text file .

2. Each entry must be separated by one space only. Otherwise data read error will be generated.

NOTE : In case of any error while reading the data file ,the table editor entries will be blank .

If a numeric entry in the data file contains non numeric

characters the cell in the table editor will contain either "ERR" or "-" (depending on the size of the field) .

NAVIGATION :

The table editor works in two modes i.e navigation and editing modes . In the navigation mode, the columns and rows can be explored using arrow keys. A highlighted bar shows the selected cell . To edit the column the focus must be moved to the particular column and then press <ENTER> to enter data ,after editing press<ENTER> to abort the editing. In the edit mode the cell entries can be altered , in the edit mode the DEL , BACKSPACE keys have there usual function . To shift from one component to another in the table editor , use <TAB> .If the column is locked the highlighted bar goes red. Normally it is pink in color .

THE OTHER API'S

Setting the component state :

The following API is used to activate or deactivate the particular component in the window ,

```
void Set_Component_State(unsigned id , unsigned pid , unsigned short procid , short state);
```

Where the parameters are

- id refers to the identifier value of the component .
- pid refers to the identifier value of the parent window in which the component is created.
- procid is the value returned by GUIlogin( ).
- If state = 0 then the component in activated or it is deactivated.

Example :

```
Set_Component_State(BUTTON_1,WINDOW_1,hproc,1);
```

Here , the button whose id is BUTTON_1 is deactivated in the window whose id is WINDOW_1.

Deleting a window component :

The API below delete the particular component whose identifier value is given ,

short Delete_Window_Component(unsigned id , unsigned pid , unsigned short procid);
Where the parameters are

- id refers to the identifier value of the component .
- pid refers to the identifier value of the parent window in which the component is created.
- procid is the value returned by GUIlogin( ).

Example :

Delete_Window_component(BUTTON_1,WINDOW_1,hproc);

Button whose id is BUTTON_1 is deleted in the window .

Setting focus to a component :

The following API is used to set the focus to the component whose id is specified ,

short Set_Focus_To(unsigned id , unsigned pid , unsigned short procid);

Where the parameters are

- id refers to the identifier value of the component .
- pid refers to the identifier value of the parent window in which the componnet is created.
- procid is the value returned by GUIlogin( ).

Example :

Set_Focus_To(BUTTON_1,WINDOW_1,hproc);

The button whose id is BUTTON_1 is set to focus in the window whose id is WINDOW_1.

Getting the focused component :

It returns the identifier value of the currently focused component in the window,

```
Get_Focussed_Component( unsigned  pid , unsigned  procid);
```

Where the parameters are

- pid refers to the identifier value of the parent window in which the components are created.
- procid is the value returned by GUIlogin( ).

Example :

```
Get_Foccused_Component(WINDOW_1,hproc);
```

It get the focused components in the window .

Getting the next component information :

It returns the identifier value of the component next to the component "id" .

```
short   Get_Next_Component_Info(unsigned id , unsigned pid ,unsigned
short  procid , unsigned *nextid , unsigned *nexttype);
```

Where the parameters are

- id refers to the identifier value of the component .
- pid refers to the identifier value of the parent window in which the componnet is created.
- procid is the value returned by GUIlogin( ).
- The two fields are used to get the next id and type. The last field is reserved.