User Interface Objects

Constructing an application starts by deciding the use of different UI resources. There are many different UI resources in our system that can be used for designing different layouts of the applications. The following table lists all UI resources provided by our system. In this chapter, the resources are introduced. The behavior, different styles, user-input parameters, comments and event flows are presented.

1. FORM

1a. Characteristics and Behavior

FORM resource is a container-type object. It acts like a container to contain other UI objects. The UI objects, which are on the form, can be placed in all positions within the bounds of their corresponding form object. Form resource can be viewed as one page of an application and it is the basic unit in the structure of an application. By default, FORM resources contain four different types: Background, Normal, Bitmap dialog and Non-bitmap dialog. Base on the requirement of the applications to select one of the types.

1b. Event Flow

Form is the base of a number of UI objects in a form. All the UI events are passed to the *FormHandleEvent* that is the function to handle events in a form before passing to the specified object for further processing.

If an EVT_PEN_DOWN is received within the bounds of the active form, then the *FormHandleEvent* will check whether the pen goes down onto an object. If it is, then the event handler of the corresponding object is called to handle the EVT_PEN_DOWN.

If an EVT_PEN_MOVE or EVT_PEN_UP is received within the bounds of the active form, then *FormHandleEvent* will pass the event to the handler function of the object that has received the EVT_PEN_DOWN in previous time.

All other events, such as EVT_CONTROL_ENTER, EVT_LIST_ENTER and EVT_MENU_EXIT, are all passed directly to corresponding event handler function for further process.

1c. Data Structure

```
struct Form_Objects_List
{
        ObjectID            object_id;
        BYTE                object_type;
};
typedef struct Form_Objects_List FormObjectsList;

struct Form_Attr
{
        BOOLEAN             form_drawn;
        BOOLEAN             form_active;
        BOOLEAN             form_updated;
        BOOLEAN             form_save_behind;
};
typedef struct Form_Attr FormAttr;

/* FORM Structure */
struct _Form
{
        Identification      identification;
        ObjectBounds        bounds;
        ObjectID            form_focus;
        BYTE                form_style;
        BYTE *              form_dialog_title;
        BitmapTemplate      form_bitmap;
        USHORT              form_num_objects;
        FormObjectsList *   form_objects_list;
        BitmapTemplate      save_behind;
        FormAttr            form_attr;
};
typedef struct _Form Form;
```

The following table shows all the parameters of the relative UI objects and discusses the function of them.

Parameters	*Function*
identification	Information to identification the object: • ui_object_id is the object ID of the object. • ui_object_type is the object type FORM, CONTROL, FIELD, LIST, MENU, SCROLLBAR, TABLE, STRING, BITMAP and LINE.

bounds	Screen relative coordinates of the top left corner of the form. The bounds field contains the following parameters: xcoord — Screen relative X-coordinate of the object. ycoord — Screen relative Y-coordinate of the object. width — Width of the object. height — Height of the object.
form_focus	Object ID of the relative object that has the focus in the particular form.
form_style	Style of the particular Form. For example: BACKGROUND ~ Background Form without any pre-designed template. NORMAL_FORM ~ A 160 x 160 background with default template. BITMAP_DIALOG ~ Dialog with pre-design bitmap diagram on the upper left corner. NON_BITMAP_DIALOG ~ DIALOG without pre-design diagram.
form_dialog_title	Pointer to the title of the form. If text is NULL, the particular Form is no label. Only Normal Form, Bitmap and Non-Bitmap Dialog have text labels at the top of the Form.
form_bitmap	Pointer to the Bitmap structure of the Form. Normal Form and Bitmap Dialog have a predefined Bitmap diagram on the upper left corner of the From.
form_num_objects	Number of UI object contained in the Form.
form_objects_list	Pointer to an array of objects contained in the Form. The structure contains the following parameters: object_id — Object ID of the Form, specified by the resource compiler. object_type — The type of the object within the Form. For example: CONTROL, FIELD, LIST, MENU, SCROLLBAR, TABLE, STRING, BITMAP, LINE TEXTBOX.
save_behind	Used to store the information of the Bitmap covered by the specified object.

	The structure contains the following parameters:	
	xcoord	Screen relative X-coordinate of the top left corner of the covered image.
	ycoord	Screen relative Y-coordinate of the top left corner of the covered image.
	width	Width of the covered image.
	height	Height of the covered image.
	compresed	Indicate whether the Bitmap is compressed or not.
	quantisation	Number of quantization levels per pixel.
	size	Size (No. of bytes) of the Bitmap.
	bitmap_data	Pointer to the locations of storing the covered image.
form_attr	Attribute of the Form object. The from_attr field contains the parameters form_drawn, form_active, form_updated, form_save_behind.	
	form_drawn	indicate whether the Form is drawn on screen or not.
	from_active	indicate whether the Form is being used or not.
	from_updated	indicate whether the Form is being updated or not.
	form_save_behind	indicate whether other UI object is covered by the Form or not.

1d. API Functions

The following API functions can be used to manipulate form object.

- FormAddOneObject
- FormCheckObjectExists
- FormCheckStyle
- FormDeleteAllFormObjects
- FormDeleteForm
- FormDispatchEvent
- FormDrawForm
- FormEraseForm
- FormGetActiveFormID
- FormGetActiveObject
- FormGetControlValue
- FormGetMenuID
- FormGetNumberOfObjects

- FormGetObjectBounds
- FormGetObjectPointer
- FormInitAllFormObjects
- FormInitForm
- FormPopupForm
- FormSetActiveForm
- FormSetControlGroupSelection
- FormSetDialogTitle
- FormSetEventHandler
- FormSetFormActiveObject
- FormSetObjectBounds
- FormSetObjectPosition

2. BITMAP

2a. Characteristics and Behavior

Bitmap object can be used for 2 purposes:

- predefined bitmap
- button with bitmap on it

For the use as a bitmap, the bitmap object is disabled and any pen action will not be passed to the event handler function of the bitmap object.

For the use as a button, when the user press on the bitmap object with the pen, the bitmap object either highlights or change to another bitmap (depends on the style of the bitmap object) until the pen is lifted or is moved out of the bounds of the bitmap object.

2b. Event Flow

There are 6 events for are handled by the *BitmapHandleEvent*. They are:

- EVT_PEN_UP
- EVT_PEN_DOWN
- EVT_PEN_MOVE
- EVT_BITMAP_ENTER
- EVT_BITMAP_EXIT
- EVT_BITMAP_SELECT

Among those events, EVT_BITMAP_ENTER, EVT_BITMAP_EXIT and EVT_BITMAP_SELECT are all generated by *BitmapHandleEvent*. The structures of the events can be found in the document of UI Events.

Events that are passed to BitmapHandleEvent	***Actions to be taken by BitmapHandleEvent***
EVT_PEN_DOWN	At the start, the bitmap object is in IDLE state. If the position of the pen is within the bounds, then EVT_BITMAP_ENTER is sent.
EVT_BITMAP_ENTER	The bitmap object is now in ENTER state. The bitmap object is highlighted or another bitmap is pasted on it.
EVT_PEN_MOVE	If it is now in ENTER state, ➢ if the pen is moved out of the bounds of the bitmap object, then EVT_BITMAP_EXIT is sent. ➢ if the pen is still within the bounds, then nothing happens. If it is now in EXIT state, ➢ if the pen is moved back to and within the bounds of the bitmap object, then EVT_BITMAP_ENTER is sent. ➢ if the pen is still outside the bounds, then nothing happens.
EVT_BITMAP_EXIT	The bitmap object is now in EXIT state. The display of the bitmap object is changed back to original one.
EVT_PEN_UP	If it is still ENTER state now, then EVT_BITMAP_SELECT is sent and the display of the bitmap object is back to normal. But on the other hand, if it is EXIT state now, then state is changed back to IDLE state.

2c. Data Structure

```
struct Bitmap_Attr
{
        BOOLEAN         bitmap_drawn;
        BOOLEAN         bitmap_enable;
        BOOLEAN         bitmap_active;
        BOOLEAN         bitmap_enter;
        BOOLEAN         bitmap_visible;
};
```

```
typedef struct Bitmap_Attr BitmapAttr;

/* BITMAP Structure */
struct _Bitmap
{
        Identification          identification;
        BitmapTemplate          bitmap_bitmap1;
        BitmapTemplate          bitmap_bitmap2;
        ObjectBounds            bounds;
        ObjectBounds            screen_bounds;
        BYTE                    bitmap_style;
        BitmapAttr              bitmap_attr;
 };
typedef struct  _Bitmap Bitmap;
```

The following table shows all the parameters of the relative UI objects and discusses the function of them.

Parameters	*Function*
identification	Information to identification the object: • ui_object_id is the object ID of the object. • ui_object_type is the object type
bitmap_bitmap1	Pointer to the Bitmap structure of the original Bitmap diagram.
bitmap_bitmap2	Pointer to the Bitmap structure of the secondary Bitmap diagram. If the original Bitmap is clicked, the secondary Bitmap will instead of the original and place on the screen.
bounds	If the object is pasted within a table, then the bounds field represents the **Table** relative coordinates of the top left corner of the Bitmap. Otherwise represents the **Screen** relative coordinates of the top left corner of the Bitmap. The bounds field contains the following parameters: xcoord Table / Screen relative X-coordinate of the object. ycoord Table / Screen relative Y-coordinate of the object. width Width of the object. height Height of the object. Table relative means the bounds is related to the specified table object and the top left corner of bounds is relative to the top left corner of the table cell that holds the bitmap object. These parameters will only be concerned when the particular Bitmap object is paste into a table.

screen_bounds	Screen relative coordinates of the top left corner of the Bitmap. It means that the top left corner of the bitmap object is relative to the top left corner of the screen. The bounds field contains the following parameters: xcoord — Screen relative X-coordinate of the object. ycoord — Screen relative Y-coordinate of the object. width — Width of the object. height — Height of the object.
bitmap_style	Style of the particular Bitmap. For example: BITMAP_STYLE_0 ~ When the bitmap is clicked, the bitmap is Inverted. BITMAP_STYLE_1~ When the bitmap is clicked, another is pasted on it.
bitmap_attr	Attribute of the Bitmap object. The bitmap_attr field contains the parameters bitmap_drawn, bitmap_enable, bitmap_active, bitmap_enter, bitmap_visible. bitmap_drawn — indicate whether the Bitmap is drawn on screen or not. bitmap_enable — indicate whether the Bitmap response to the pen action. bitmap_active — indicate whether the Bitmap is being used or not. bitmap_enter — indicate whether the Bitmap is being clicked or not. bitmap_visible — indicate whether the Bitmap is visible on screen or not.

2d. API Functions

The following API functions can be used to manipulate bitmap object.

- BitmapDeleteBitmap
- BitmapDrawBitmap
- BitmapEraseBitmap
- BitmapGetAttribute
- BitmapGetBitmapBounds
- BitmapGetBitmapTemplate

- BitmapInitBitmap
- BitmapSetAttribute
- BitmapSetBitmapBounds
- BitmapSetBitmapTemplate

3. CONTROL

3a. Characteristics and Behavior

Control resource is a family of different types of UI resources in which there are 5 different types of control objects. They are *button, repeat button, checkbox, popup trigger and push button.*

Button is a simple user interactive object that is existing in many operating systems. By simple clicking on it, the button can trigger event in an application. When it is being pressed, the button is inverted in color until the pen is lifted or the pen is moved out of the bounds of the button. A button displays as a text label surrounded by a rectangular frame. The frame has rounded corners with text has regular size. The text can be selected displays at the left, right or in the middle of the frame by the setting of the control_text_alignment attribute. If the text is too long to be fully displayed in the frame, the characters of the text will be automatically minimized to make room to fit the remainder into the frame. Dot sign will then be added to the end of the remained characters.

Repeat button is also a button-type object. The difference between button and repeat button is that user can trigger the button repeatedly by pressing on it continuously. When it is being pressed, it is also inverted in color until the pen is not within the bounds of the repeat button object. A repeat button is an arrow or an arrow in the middle and surrounded by a rectangular frame. The frame has rounded corners. A good example for a repeating button is the scroll arrow.

Popup Trigger is a button object. But when it is selected, then a pop up list of items is displayed on the screen for users' further selection. After selection, the pop up list is erased and the original button-like popup trigger is redrawn again. A popup trigger displays as a text label surrounded by a rectangular frame and an upside-down triangle on the right hand side of the label. The frame has rounded corners with text has regular size. The text can be selected displays at the left, right or in the middle of the frame depends on the setting of the control_text_alignment attribute. If the text is too long to be fully displayed in the frame, the characters of the text will be automatically minimized to make room to fit the remainder into the frame. Dot sign will add to the end of the remained characters.

Checkbox provides the ability for an application to show the checking of an option. Checkbox has a setting – checked or unchecked. When the checkbox is selected once, the setting of that checkbox is toggled from checked to unchecked or from unchecked to checked according to the initial states of the setting. The state of the checkbox is present as two different bitmaps. One is for checked state and the other is for unchecked state. Check box is a small square control object and always appears with an optional text label beside the box.

Push Button has the same behavior of the checkbox. But when push button is toggled, the color of the push button is inverted. Push buttons display a text label surrounded by a rectangular frame. The frame may have rounded corners depend on the subtype setting. Text with regular size and if it is too long to be fully displayed in the frame, the characters of the text will be automatically minimized to make room to fit the remainder into the frame. Dot sign will add to the end of the remained characters. When push buttons appear in a horizontal or vertical row with no pixels between the buttons. The buttons share a common border so there appears to be one pixel line between two push buttons.

Both the checkboxes and push buttons can be grouped together in their own kind and distinguished by a unique ID number. In this mode of operation, only one of the checkbox or push button can be checked at the same time.

3b. Event Flow

There are 8 events that are handled by the *ControlHandleEvent*. They are:

- EVT_PEN_UP
- EVT_PEN_DOWN
- EVT_PEN_MOVE
- EVT_CONTROL_ENTER
- EVT_CONTROL_EXIT
- EVT_CONTROL_REPEAT
- EVT_CONTROL_SELECT
- EVT_CONTROL_POP_SELECT

Among those events, EVT_CONTROL_ENTER, EVT_CONTROL_EXIT, EVT_CONTROL_REPEAT, EVT_CONTROL_SELECT and EVT_BITMAP_SELECT and EVT_CONTROL_POP_SELECT are all generated by *ControlHandleEvent*. The structures of the events can be found in the document of UI Events.

Events that are passed to ControlHandleEvent	*Actions to be taken by ControlHandleEvent*

EVT_PEN_DOWN	At the start, the control object is in IDLE state. If the position of the pen is within the bounds, then EVT_CONTROL_ENTER is sent. If it is now in POPUP state and the position of the pen is within one of the items in the popup list, then EVT_CONTROL_ENTER with the highlighted item is sent.
EVT_CONTROL_ENTER	The control object is now in ENTER state. The control object is highlighted or another bitmap is pasted on it. It depends on the characteristics of the type of control object. If it is a repeat button, then EVT_CONTROL_REPEAT is sent and the timer for counting is reset. If it is a popup trigger and the popup list is displayed, then the item on the first pen down is highlighted.
EVT_PEN_MOVE	If it is now in ENTER state, ➢ if the pen moves out of the bounds of the control object (or the bounds of the item on the first pen down in the popup list if the control object is popup trigger), then EVT_CONTROL_EXIT is sent. ➢ if the pen is still within the bounds of the control object and it is a repeat button, then if the timer reaches a predefined value, EVT_CONTROL_REPEAT is sent again. Otherwise, nothing happens. ➢ if the pen is still within the bounds of the control object (or the bounds of the item on the first pen down in the popup list if the control object is popup trigger) and it is not a repeat button, then nothing happens. If it is now in EXIT state, ➢ if the pen moves back to and within the bounds of the control object (or the bounds of the item on the first pen down in the popup list if the control object is popup trigger), then EVT_CONTROL_ENTER is sent. ➢ if the pen is still outside the bounds (or the bounds of the item on the first pen down in the popup list if the control object is popup trigger), then nothing happens.
EVT_CONTROL_EXIT	The control object is now in EXIT state. The display of the control object is changed back to original one. If it is a

	popup trigger, the highlighted item will change back to original status.
EVT_PEN_UP	If it is in ENTER state and it is a popup trigger control object, then EVT_CONTROL_POP_SELECT is sent with the item number of the selected item. The popup list is erased and the original popup trigger object is redrawn. If it is still ENTER state and it is not a popup trigger, then EVT_CONTROL_SELECT is sent and the display of the control object is back to normal. But on the other hand, if it is EXIT state now, then state is changed back to IDLE state of the control object.
EVT_CONTROL_SELECT	If it is a popup trigger, then popup list is displayed and the control object is back to IDLE state.

3c. Data Structure

```
struct Control_Attr
{
        BOOLEAN         control_enable;
        BOOLEAN         control_drawn;
        BOOLEAN         control_save_behind;
        BOOLEAN         control_active;
        BOOLEAN         control_enter;
        BOOLEAN         control_enter1;
        BOOLEAN         control_enter2;
        BOOLEAN         control_visible;
};
typedef struct Control_Attr ControlAttr;

/* Structure: Control_Template_Button */
struct Control_Template_Button
{
        USHORT          button_radius;
        BYTE            button_color_on;
        BYTE            button_color_off;
        BOOLEAN         control_value;
};
typedef struct Control_Template_Button ControlTemplateButton;



/* Structure: Control_Template_PushButton */
struct Control_Template_PushButton
```

```
{
        BYTE            push_button_color_on;
        BYTE            push_button_color_off;
        USHORT          push_button_group_id;
        BOOLEAN         control_value;
        USHORT          push_button_radius;
};
typedef struct Control_Template_PushButton ControlTemplatePushButton;

/* Structure: Control_Template_RepeatButton */
struct Control_Template_RepeatButton
{
        USHORT          repeat_count;
        BitmapTemplate  repeat_bitmap;

};
typedef struct Control_Template_RepeatButton ControlTemplateRepeatButton;

/* Structure: Control_Template_CheckBox */
struct Control_Template_CheckBox
{
        BitmapTemplate  checkbox_bitmap1;
        BitmapTemplate  checkbox_bitmap2;
        USHORT          checkbox_group_id;
        BOOLEAN         control_value;
};
typedef struct Control_Template_CheckBox ControlTemplateCheckBox;

/* Structure: Control_Template_Popup_Trigger */
struct Control_Template_Popup_Trigger
{
        USHORT          popup_radius;
        USHORT          popup_num_objects;
        BYTE **         popup_items_list;
        BitmapTemplate  save_behind;
        ObjectBounds    popup_arrow_down_bounds;
        SHORT           popup_selected_item;
        SHORT           popup_highlighted_item;
        USHORT          popup_max_num_item_display;
        USHORT          popup_current_num_item_display;
        USHORT          popup_top_item_num;
        ObjectBounds    popuped;
        BOOLEAN         control_value;
        BOOLEAN         popup_arrow_up;
        BOOLEAN         popup_arrow_down;
```

```
};
typedef struct Control_Template_Popup_Trigger ControlTemplatePopupTrigger;

/* Structure: _Control */
struct _Control
{
        Identification      identification;
        BYTE *              control_text;
        ObjectBounds        bounds;
        void *              control_template;
        BYTE                control_style;
        BYTE                control_subtype;
        ControlAttr         control_attr;
};
typedef struct _Control Control;
```

The following table shows all the parameters of the relative UI objects and discusses the function of them.

Parameters	*Function*
identification	Information to identification the object: • ui_object_id is the object ID of the object. • ui_object_type is the object type
control_text	Pointer to the label of the control. If text is NULL, the particular control is no label. Only buttons, push buttons and popup trigger have text labels.
bounds	If the object is pasted within a table, then the bounds field represents the **Table** relative coordinates of the top left corner of the Control object. Otherwise represents the **Screen** relative coordinates of the top left corner of the Control object. The bounds field contains the following parameters: xcoord Table / Screen relative X-coordinate of the object. ycoord Table / Screen relative Y-coordinate of the object. width Width of the object. height Height of the object.
screen_bounds	Screen relative coordinates of the top left corner of the Control. The bounds field contains the following parameters: xcoord Screen relative X-coordinate of the object. ycoord Screen relative Y-coordinate of the object. width Width of the object.

	height Height of the object.
control_template	Pointer to the template structure of the selected Control object. The control_template may pointed to one of the following template structure: • *Control_Template_Button* The Control_Template_Button contains the following parameters: button_radius The radius of the button edge. (This value should set to 2 for gray design) button_color_on The color of the button when it is being pressed. button_color_off The color of the button when it is not being pressed. control_value An On / Off value to show whether the button is pressed or not. • *Control_Template_PushButton* The Control_Template_PushButton contains the following parameters: push_button_color_on The color of the push button when it is being pressed. push_button_color_off The color of the push button when it is not being pressed. push_button_group_id The group ID of the push buttons control_value An On / Off value to show whether the push button is pressed or not. push_button_radius The radius of the push button edge. (This value should set to 2 for gray design) • *Control_Template_RepeatButton* The Control_Template_RepeatButton contains the following parameters: repeat_count A number to show when to send a EvtControlRepeatEvent. repeat_bitmap Pointer to the Bitmap structure of the repeat button. • *Control_Template_CheckBox* The Control_Template_CheckBox contains the following parameters:

checkbox_bitmap1	Pointer to the Bitmap structure of the check box.
checkbox_bitmap2	Pointer to the Bitmap structure of the check box.
checkbox_button _group_id	The group ID of the check box.
control_value	An On / Off value to show whether the check box is pressed or not.

- *Control_Template_Popup_Trigger*

The Control_Template_Popup_Trigger contains the following parameters:

popup_radius	The radius of the push button edge. (This value should set to 2 for gray design)
popup_num_objects	Number of items in the list.
popup_items_list	Pointer pointed to the list of items in the popup list.
save_behind	Pointer pointed to the structure to store the image that is being covered.
popup_arrow_up _bounds	Screen relative bounds of the upper left corner of the popup up arrow.
popup_arrow_down _bounds	Screen relative bounds of the lower left corner of the popup down arrow.
popup_selected _item	Item number of the selection.
popup_highlighted _item	The number of the highlighted item.
popup_max_num _item _display	Maximum number of items displayed on the list.
popup_current_num _item _display	Number of items displayed on the list.
popup_ top_ item _num	Top item number of the popup list.
popuped	Pointer pointed to the structure to store the region covered by the popup list.
control_value	An On / Off value to show whether the popup trigger is pressed or not.
popup_arrow_up	Set this attribute to indicate the popup up arrow is drawn on screen. (This attribute need to be set, once the list cannot display all items on the screen and more items contain in the previous page of the list.)
popup_arrow_down	Set this attribute to indicate the popup down arrow is drawn on screen. (This attribute need to be set, once the list cannot

		display all items on the screen and more items contain in the next page of the list.)
control_style	Style of the particular Control. For example:	BUTTON, PUSH_BUTTON, REPEAT_BUTTON, CHECKBOX, POPUP_TRIGGER
control_subtype	Pattern of the template of the control object.	
text_alignment	Alignment of the text in the frame. For example:	LEFT_ALIGN, CENTRE_ALIGN, RIGHT_ALIGN
control_attr	Attribute of the Control object. The control_attr field contains the parameters control_enable, control_drawn, control_save_behind, control_active, control_enter, control_enter1, control_enter2, control_visible.	
	control_enable	Indicate whether the Control response to the pen action.
	control_drawn	Indicate whether the Control is drawn on screen or not.
	control_save_behind	Indicate whether other UI object is covered by the Control or not.
	control_active	Indicate whether the Control is being used or not.
	control_enter	Indicate whether the Control is being clicked or not.
	control_enter1	Indicate whether the Popup up arrow is being clicked or not.
	control_enter2	Indicate whether the Popup down arrow is being clicked or not.
	control_visible	Indicate whether the Control is visible on screen or not.

3d. API Functions

The following API functions can be used to manipulate control object.

- ControlDeleteControl
- ControlDrawControl
- ControlEraseControl
- ControlGetAttributes

- ControlGetCheckedCheckbox
- ControlGetLabel
- ControlGetPushedPushButton
- ControlHighlightOneItem
- ControlHitControl
- ControlInitControl
- ControlPopupDeleteAllItems
- ControlPopupDeleteItem
- ControlPopupFindItemNum
- ControlPopupGetCurrentNumOfDisplayedItems
- ControlPopupGetPopupItem
- ControlPopupGetSelectedItem
- ControlPopupGetTopItemNumber
- ControlPopupGetTotalItems
- ControlPopupInsertItem
- ControlPopupPopupTrigger
- ControlPopupSetSelectedItem
- ControlPopupSetTotalItems
- ControlPopupTriggerClosePopupTrigger
- ControlSetAttributes
- ControlSetLabel
- ControlSetPopupScroll
- ControlUpdatePopupTrigger

4. FIELD

4a. Characteristics and Behavior

Field resource provides the ability for the application to input editable text. The text, which is in the field object, can be displayed with multiple lines. There are also many features that are implemented and supported by the field object.

They are:

- Drag-scrolling and Drag-selection
- CUT, PASTE and COPY
- Special keys – PAGE UP, PAGE DOWN, HOME, END and BACKSPACE
- Scrolling by scrollbar
- Indication of insertion point
- Switching between viewing mode and editing mode
- Protection by setting maximum number of characters

In order to give flexibility to application to filter the characters to a field object, the routine for adding key-in character to the string of field object is separated from the *FieldHandleEvent.* Therefore, the EVT_KEY with the visible character is sent to application layer for further process and the *FieldHandleEvent* would not handle EVT_KEY with visible character.

4b. Event Flow

There are 9 events that are handled by the *FieldHandleEvent*. They are:

- EVT_PEN_UP
- EVT_PEN_DOWN
- EVT_PEN_MOVE
- EVT_FIELD_ENTER
- EVT_FIELD_SELECT
- EVT_FIELD_CHANGED
- EVT_FIELD_MODIFIED
- EVT_FIELD_JOT_PASTE_STRING
- EVT_KEY

Among those events, *FieldHandleEvent* sends all EVT_FIELD_ENTER, EVT_FIELD_CHANGED, EVT_FIELD_MODIFIED and EVT_FIELD_SELECT. *JotHandleEvent* sends EVT_FIELD_JOT_PASTE_STRING to request pasting symbol into a field object. The structures of the events can be found in the document of UI Events.

Events that are passed to FieldHandleEvent	*Actions to be taken by FieldHandleEvent*
EVT_PEN_DOWN	At the start, the field object is in IDLE state. If the position of the pen is within the bounds, then EVT_FIELD_ENTER is sent.
EVT_FIELD_ENTER	The field object is now in ENTER state. The insertion point of the field object is set to display on the position of the pen.
EVT_PEN_MOVE	If it in now in ENTER state, ➢ when the pen moves around, the text between the position of the first pen down and the current position of the pen is highlighted. After highlighting, the insertion point disappears. ➢ if the text in the field object is required to scroll because of the drag-selection, then EVT_FIELD_CHANGED is sent.

EVT_PEN_UP	If it is in ENTER state, then when the pen is lifted, EVT_FIELD_SELECT is sent. The field object is back to IDLE state and the highlighted section of text is still on the field object.
EVT_KEY with BACKSPACE	The character on the left of the insertion point is deleted. EVT_FIELD_CHANGED is sent.
EVT_KEY with CUT	The text that is being highlighted is cut to clipboard. EVT_FIELD_CHANGED is sent.
EVT_KEY with COPY	The text that is being highlighted is copied to clipboard.
EVT_KEY with PASTE	The text in clipboard is pasted to the field object at the insertion point or to replace the highlighted section in the field object. EVT_FIELD_CHANGED is sent.
EVT_KEY with PAGE UP	The text in the field object is scrolled up for one page and the insertion point is re-positioned.
EVT_KEY with PAGE DOWN	The text in the field object is scrolled down for one page and the insertion point is re-positioned.
EVT_KEY with HOME	The first line of text in the string of the field object is displayed at the top of the field object and the insertion point is positioned in the top-left corner of the field object.
EVT_KEY with END	The last line of text in the string of the field object is displayed at the bottom of the field object and the insertion point is positioned in the bottom-right corner of the field object.
EVT_KEY with UP ARROW	The insertion point is moved up one line.
EVT_KEY with DOWN ARROW	The insertion point is moved down one line.
EVT_KEY with LEFT ARROW	The insertion point is moved to the left for one character position. If the current position of the insertion point is already in the left margin, then the insertion point will be moved to the end of the previous line.
EVT_KEY with RIGHT ARROW	The insertion point is moved to the right for one character position. If the current position of the insertion point is

already in the right margin, then the insertion point will be moved to the start of next line.

4c. Data Structure

```
/* Structure: Line Info */
struct _LineInfo
{
        WORD            start;
        WORD            length;
};
typedef struct _LineInfo LineInfo;

/* Structure: Field_Underlined_Section */
struct _FieldUnderlinedSection
{
        WORD            field_underlined_start_char;
        WORD            field_underlined_length;
};
typedef struct _FieldUnderlinedSection FieldUnderlinedSection;

/* Structure: Field_Attr */
struct Field_Attr
{
        BOOLEAN         field_drawn;
        BOOLEAN         field_active;
        BOOLEAN         field_enable;
        BOOLEAN         field_dirty;
        BOOLEAN         field_highlight;
        BOOLEAN         field_insert_pt_visible;
        BOOLEAN         field_scrollbar;
        BOOLEAN         field_full_size;
        BOOLEAN         field_visible;
};
typedef struct Field_Attr FieldAttr;

/* Structure: _Field */
struct _Field
{
        Identification  identification;
        ObjectBounds    bounds;
        ObjectBounds    screen_bounds;
        BYTE *          field_string;
        BYTE            field_style;
```

```
        BYTE            field_back_line;
        BYTE            field_font_id;
        BYTE            field_font_color;
        BYTE            field_background_color;
        BYTE            field_text_alignment;
        WORD            field_max_chars;
        WORD            field_current_num_chars;
        WORD            field_total_num_lines;
        WORD            field_top_line_num;
        WORD            field_num_lines_displayed;
        SHORT           field_insert_pt_x;
        SHORT           field_insert_pt_y;
        WORD            field_insert_pt_char_pos;
        BYTE            field_insert_pt_movement;
        WORD            field_highlight_start_char;
        WORD            field_highlight_end_char;
        WORD            field_highlight_length;
        USHORT          field_num_underlined_section;
        FieldUnderlinedSection **field_underlined_section;
        LineInfo *      field_lineinfo;
        USHORT          field_repeat_count;
        FieldAttr       field_attr;
};
typedef struct _Field Field;
```

The following table shows all the parameters of the relative UI objects and discusses the function of them.

Parameters	*Function*
identification	Information to identification the object: • ui_object_id is the object ID of the object. • ui_object_type is the object type
bounds	If the object is pasted within a table, then the bounds field represents the **Table** relative coordinates of the top left corner of the Field object. Otherwise represents the **Screen** relative coordinates of the top left corner of the Field object. The bounds field contains the following parameters: xcoord Screen / Table relative X-coordinate of the object. ycoord Screen / Table relative Y-coordinate of the object. width Width of the object. height Height of the object.
screen_bounds	Screen relative coordinates of the top left corner of the Field. The

	bounds field contains the following parameters: xcoord Screen relative X-coordinate of the object. ycoord Screen relative Y-coordinate of the object. width Width of the object. height Height of the object.
field_string	Pointer pointed to the text string in the field object.
field_style	Style of the particular Field. For example: FIELD_STYLE_0 = No frame, FIELD_STYLE_1 = frammed.
field_back_line	Style of the lines under the field's text. For example: NO_LINE ~ Without lines under the text. DOT_LINE ~ With dot lines under the text. GRAY_LINE ~ With gray lines under the text. (The color of the lines is predefined.)
field_font_id	Font type of the field's text. For example: SMALL_FONT, MEDIUM_FONT and LARGE_FONT
field_font_color	Color of the field's text. For example: COLOR_WHITE, COLOR_GREY1, COLOR_GREY2, COLOR_BLACK
field_background_color	Background color of the field. For example: COLOR_WHITE, COLOR_GREY1, COLOR_GREY2, COLOR_BLACK
field_text_alignment	Alignment of the text in the field. For example: LEFT_ALIGN, CENTRE_ALIGN, RIGHT_ALIGN
field_max_chars	Maximum number of characters in the field object.
field_current_num_chars	Current number of characters in the string displayed by the field object; the null-terminator is excluded.
field_total_num_lines	Total number of lines in the field object.
field_top_line_num	Top line number in the displayed field.
field_num_lines_displayed	Number of lines displayed in the field object.

field_insert_pt_x	Column position of the insertion point.
field_insert_pt_y	Row position of the insertion point.
field_insert_pt_ char_pos	The character poistion beside the insert point.
field_insert_pt_ movement	Movement direction in the field. For example: NO_MOVEMENT, MOVE_UP, MOVE_DOWN, MOVE_LEFT, MOVE_RIGHT
field_highlight_ start_char	Starting character position of the current selection.
field_highlight_ end_char	Ending character position of the current selection.
field_highlight_ length	Length of the current selection. If the *field_highlight_start_char* is equal to the *field_highlight_end_char*, there is no selection.
field_num_ underlined_section	Number of lines is being selected.
field_underlined_ section	Pointer to an arrau of the FieldUnderlinedSection structure. This structure contains the following parameters: field_underlined_start_char — The character position of the first character being underlined. field_underlined_length — The number of characters being underlined.
field_lineinfo	Pointer to an array of the LineInfo structures. This structure contains the following parameters: start — The character position of the first character displayed by a line. length — The number of characters displayed by a line.
field_repeat_count	Number of lines repeated displays in the field object.
field_attr	Attribute of the Field object. The field_attr field contains the parameters field_drawn, field_active, field_enable, field_dirty, field_highlight, field_insert_pt_visible, field_scrollbar, field_full_size, field_visible.

field_drawn	Indicate whether the Field is drawn on screen or not.
field_active	Indicate whether the Field is being used or not.
field_enable	Indicate whether the Field response to the pen action.
field_dirty	Indicate the field object has been changed. (e.g. cut, paste)
field_highlight	Indicate whether the field is highlighted or not.
field_insert_pt _visible	Indicate whether the insert point of the field is visible or not.
field_scrollbar	Indicate the particular field has a scrollbar.
field_full_ size	Indicate the particular field with full size of the table object.
field_visible	Indicate whether the Field is visible on screen or not.

4d. API Functions

The following API functions can be used to manipulate field object.

- FieldAddKeyInChar
- FieldCharPosToLineNum
- FieldDeleteField
- FieldDeleteString
- FieldDirty
- FieldDrawField
- FieldEraseField
- FieldGetAttribute
- FieldGetCurrentHighlightedSelection
- FieldGetDisplayRowNum
- FieldGetFieldBounds
- FieldGetFirstVisibleChar
- FieldGetFont
- FieldGetInsertPointPosition
- FieldGetLastVisibleChar
- FieldGetMaxNumChars
- FieldGetMaxNumLinesDisplay
- FieldGetNumBlankLines
- FieldGetNumOfChars
- FieldGetNumOfLinesDisplayed

- FieldGetScrollbarAttribute
- FieldGetTextPointer
- FieldGetTopLineNum
- FieldGetTotalNumOfLines
- FieldInitField
- FieldInsertString
- FieldPasteString
- FieldScrollField
- FieldSetAttribute
- FieldSetBounds
- FieldSetDirty
- FieldSetFont
- FieldSetHighlightSelection
- FieldSetInsertPointOff
- FieldSetInsertPointOn
- FieldSetInsertPointPositionByCharPos
- FieldSetInsertPointPositionByXY
- FieldSetMaxNumChars
- FieldSetScrollbarAttribute
- FieldSetText
- FieldSetTopLineNum
- FieldUndo

5. LINE

5a. Characteristics and Behavior

Line resource, as stated before, is an object that can provide the ability for application to draw lines on the display. The application can select different thickness, color and length of the line.

5b. Event Flow

This object is supposed to be used as a drawing on the display. Therefore, there is no event associated to this line object.

5c. Data Structure

```
struct Line_Attr
{
        BOOLEAN             line_drawn;
```

```
        BOOLEAN         line_visible;
};
typedef struct Line_Attr LineAttr;

/* Structure: _Line */
struct _Line
{
        Identification      identification;
        ObjectBounds        bounds;
        BYTE                line_color;
        BYTE                line_style;
        BYTE                line_thick;
        USHORT              line_endpt_xpos;
        USHORT              line_endpt_ypos;
        LineAttr line_attr;
};
typedef struct _Line Line;
```

The following table shows all the parameters of the relative UI objects and discusses the function of them.

Parameters	*Function*
identification	Information to identification the object: • ui_object_id is the object ID of the object. • ui_object_type is the object type
bounds	The bounds field to represent the Screen relative coordinates of the left hand side of the Line. The bounds field contains the following parameters: xcoord Screen relative X-coordinate of the object. ycoord Screen relative Y-coordinate of the object. width Width of the object. height Height of the object.
line_color	Color of the line object. For example: COLOR_WHITE, COLOR_GREY1, COLOR_GREY2, COLOR_BLACK
line_sytle	Style of the line object. For example: DOTTED_LINE, NON_DOTTED_LINE
line_thick	Thickness of the line object. For example: 1 = Single line, 2 = Double line and so on.

line_endpt_xpos	Window relative x-coordinate of the end point.
line_endpt_ypos	Window relative y-coordinate of the end point.

5d. API Functions

The following API functions can be used to manipulate line object.

- LineDeleteLine
- LineDrawLine
- LineEraseLine
- LineGetAttribute
- LineGetLineCharacteristics
- LineGetPosition
- LineInitLine
- LineSetAttribute
- LineSetLineCharacteristics
- LineSetPosition

6. LIST

6a. Characteristics and Behavior

List resource provides a way for the application to show a list of items for choices vertically. The list object also provides 2 small arrows at the positions of the right margin to the first and last visible choice. Whenever the item is being highlighted or the item is already selected, the item is inverted in color.

Generally, there are 2 types of actions that are taken by the list object. They are the pen down action on one of the visible choice and the pen down action on the small arrow. For the case that the pen is on a visible choice, when the pen is on a choice, then the original selected choice is un-highlighted and the new choice is highlighted. When the pen is moved along the list of choices, the item under the pen is highlighted instead.

For the pen down on the arrow, the list is scrolled wither up or down. When the list is scrolled up, the first visible choice of the current display will become the last choice of the scrolled list. But if the number of choices in the list is not enough to do the scrolling, then the first choice in the list object is displayed as the first visible choice on the display. It is the same case as scrolling down.

There is something that is important to be highlighted in here. Two or more list object can link together to form a large list object with several columns of visible choice. The

list objects can be arranged synchronously or asynchronously. Synchronous list objects means that when pen down is on an item of a list object, then the same item in the other list object is also highlighted at the same time. Asynchronous list objects means that there can only be one highlighted choice in the list objects. If a choice is being highlighted in a list object, all other visible choices in other list objects are un-highlighted.

6b. Event Flow

There are 7 events that are handled by the *ListHandleEvent*. They are:

- EVT_PEN_UP
- EVT_PEN_DOWN
- EVT_PEN_MOVE
- EVT_LIST_ENTER
- EVT_LIST_EXIT
- EVT_LIST_SELECT
- EVT_LIST_ARROW

Among those events, EVT_LIST_ENTER, EVT_LIST_EXIT, EVT_LIST_SELECT and EVT_LIST_ARROW are all generated by *ListHandleEvent*. The structures of the events can be found in the document of UI Events.

Events that are passed to ListHandleEvent	*Actions to be taken by ListHandleEvent*
EVT_PEN_DOWN	At the start, the list object is in IDLE state. If the position of the pen is within the bounds, then EVT_LIST_ENTER is sent.
EVT_LIST_ENTER	The list object is now in ENTER state. If the position of the pen while it is pen down is on the up arrow, then the up arrow is inverted in color. If the position of the pen while it is pen down is on the down arrow, then the down arrow is inverted in color. If the position of the pen while it is pen down is on one of the visible choice of list object, then the previous selected item is unhighlighted and the current selected choice is highlighted instead.
EVT_PEN_MOVE	If it is now in ENTER state, ➢ If up or down arrow is being highlighted and the pen is moved out of their bounds, then EVT_LIST_EXIT is sent. ➢ If one of the choices is being highlighted and the pen is moved within the bounds of the visible choices, then corresponding choice is

	highlighted. ➢ If one of the choices is being highlighted and the pen is moved out of the bounds of all visible choices, then EVT_LIST_EXIT is sent. ➢ If the pen is moved within the original highlighted region, then nothing happens. If it is now in EXIT state, ➢ If the pen is moved back to the region that was highlighted, then EVT_LIST_ENTER is re-sent again. ➢ If the pen is still out of the bounds of the region that was highlighted, then nothing happens.
EVT_LIST_EXIT	The list object is changed to EXIT state now. All the regions in the list object are unhighlighted.
EVT_PEN_UP	If it is in ENTER state and the pen is lifted, ➢ If the highlighted region is up arrow, then scrolling up is performed and the region is unhighlighted. EVT_LIST_ARROW is sent. ➢ If the highlighted region is down arrow, then scrolling down is performed and the region is unhighlighted. EVT_LIST_ARROW is sent. ➢ If the highlighted region is one of the visible choice, then the choice becomes the selection of the list object and it keeps highlighted after the pen is lifted from it. EVT_LIST_SELECT with the item number of the selected choice is sent.

6c. Data Structure

```
/* Structure: List_Attr */
struct List_Attr
{
        BOOLEAN         list_enable;
        BOOLEAN         list_drawn;
        BOOLEAN         list_active;
        BOOLEAN         list_set_scroll;
        BOOLEAN         list_enter1;
        BOOLEAN         list_enter2;
        BOOLEAN         list_visible;
```

```
        BOOLEAN         list_synchronous;
};
typedef struct List_Attr ListAttr;

/* Structure: _List */
struct _List
{
        Identification  identification;
        ObjectBounds    bounds;
        ObjectBounds    screen_bounds;
        USHORT          list_num_related_list;
        ObjectID *      list_related_list_id;
        USHORT          list_total_num_items;
        USHORT          list_max_num_items_on_display;
        USHORT          list_num_items_on_display;
        USHORT          list_top_item_num;
        BYTE **         list_items;
        SHORT           list_item_height;
        BYTE            list_text_alignment;
        SHORT           list_selected_item;
        SHORT           list_highlighted_item;
        BYTE            list_style;
        ObjectBounds    list_arrow_up_bounds;
        ObjectBounds    list_arrow_down_bounds;
        BOOLEAN         list_arrow_up;
        BOOLEAN         list_arrow_down;
        BYTE            list_text_color;
        BYTE            list_bg_color;
        BYTE            list_text_font;
        ListAttr        list_attr;
};
typedef struct _List List;
```

The following table shows all the parameters of the relative UI objects and discusses the function of them.

Parameters	*Function*
identification	Information to identification the object: • ui_object_id is the object ID of the object. • ui_object_type is the object type
bounds	If the object is pasted within a table, then the bounds field represents the **Table** relative coordinates of the top left corner of the List object. Otherwise represents the **Screen** relative coordinates of the top left corner of the List object.

	The bounds field contains the following parameters: xcoord Screen / Table relative X-coordinate of the object. ycoord Screen / Table relative Y-coordinate of the object. width Width of the object. height Height of the object.
screen_bounds	Screen relative coordinates of the top left corner of the List. The bounds field contains the following parameters: xcoord Screen relative X-coordinate of the object. ycoord Screen relative Y-coordinate of the object. width Width of the object. height Height of the object.
list_num_related_ list	Number of list related to this list object.
list_related_list_id	Pointer pointed to the Object Id of the related list object. This field is used to allow one or more list objects work together (Synchronous or Asynchronous).
list_total_num _items	Number of items in the list.
list_max_num_ items_on_display	Maximum number of items displayed in the list.
list_num_items_on_ display	Current number of items displayed in the list.
list_top_item_num	Top item number of the popup list.
list_items	Pointer to an array of pointer to the text of the choices.
list_item_height	Height of the list object.
list_text _alignment	Alignment of the text in the list. For example: LEFT_ALIGN, CENTRE_ALIGN, RIGHT_ALIGN
list_selected_item	Item number of the selection.
list_highlighted _item	The number of the highlighted item.

list_style	Style of the particular List.	
	For example: LIST_STYLE_0 ~	No frame around the list object, and the whole line will be highlighted.
	LIST_STYLE_1 ~	With predefined frame around the list object and the whole line will be highlighted.
	LIST_STYLE_2 ~	With predefined frame around the list object and only the text will be highlighted.
	LIST_STYLE_3 ~	With predefined 3D frame around the list object and the whole line will be highlighted.
	LIST_STYLE_4 ~	With predefined 3D frame around the list object and only the text will be highlighted.
	LIST_STYLE_5 ~	No frame around the list object, and the only the text will be highlighted.
	LIST_STYLE_6~	With predefined 3D frame around the list object, user can define the background color and the whole line will be highlighted.
	LIST_STYLE_7~	Specify used in the "Expense" application.
list_arrow_up_ bounds	Screen relative bounds of the upper left corner of the popup up arrow.	
list_arrow_down_ bounds	Screen relative bounds of the lower left corner of the popup up arrow.	
list_text_color	Color of the list's text.	
	For example: COLOR_WHITE, COLOR_GREY1, COLOR_GREY2, COLOR_BLACK	
list_bg_color	Background color of the list.	
	For example: COLOR_WHITE, COLOR_GREY1, COLOR_GREY2, COLOR_BLACK	
list_text_font	Font of the text.	
	For example: SMALL_FONT, MEDIUM_FONT and	

	LARGE_FONT	
list_attr	Attribute of the List object. The list_attr list contains the parameters list_drawn, list_active, list_enable, list_synchronous, list_set_scroll, list_enter1, list_enter2, list_visible.	
	list_drawn	Indicate whether the List is drawn on screen or not.
	list_active	Indicate whether the List is being used or not.
	list_enable	Indicate whether the List response to the pen action.
	list_synchronous	Indicate whether the list synchronous with another list object or not.
	list_set_scroll	Indicate whether the list contains a scrollbar or not.
	list_enter1	Indicate whether the list's up arrow is clicked or not.
	list_enter2	Indicate whether the list's down arrow is clicked or not.
	list_visible	Indicate whether the List is visible on screen or not.

6d. API Functions

The following API functions can be used to manipulate list object.

- ListDeleteAllItems
- ListDeleteItem
- ListDeleteList
- ListDrawList
- ListEraseList
- ListGetAttribute
- ListGetHighlightedItem
- ListGetListBounds
- ListGetListItem
- ListGetMaxNumItemsDisplay
- ListGetNumItemsDisplay
- ListGetNumOfItems
- ListGetSelectedItem
- ListGetTopItemNum
- ListGetTotalItems
- ListHighlightOneItem

- ListInitList
- ListInsertItem
- ListRecalculateMaxNumItemsDisplay
- ListSearchSelectedItem
- ListSetAttribute
- ListSetFont
- ListSetHighlightedItem
- ListSetListBounds
- ListSetNumItemsDisplay
- ListSetScrollList
- ListSetSelectedItem
- ListSetTopItemNum
- ListSetTotalItems
- ListUpdateList

7. MENU

7a. Characteristics and Behavior

Menu object is actually a menu bar that is displayed on the screen when the menu button is selected. The menu bar is a vertical list of choices and it is built from the bottom to the top of the screen. As the menu bar is only a single column list of choices, therefore, the total number of choices that a menu bar can handle is about 12. There is no scrolling feature implemented in the menu object. After selection of one of the menu choice, the menu bar is erased. Each application has different sets of menu names, within an application, different views may have different menus. Copy, Cut and Paste are commonly found in the menu bar.

7b. Event Flow

There are 7 events that are handled by the *MenuHandleEvent*. They are:

- EVT_PEN_UP
- EVT_PEN_DOWN
- EVT_PEN_MOVE
- EVT_MENU_ENTER
- EVT_MENU_EXIT
- EVT_MENU_SELECT_ITEM
- EVT_MENU_SELECT

Among those events, EVT_MENU_ENTER, EVT_MENU_EXIT and EVT_MENU_SELECT_ITEM are all generated by *MenuHandleEvent*. The structures of the events can be found in the document of UI Events.

Events that are passed to MenuHandleEvent	***Actions to be taken by MenuHandleEvent***
EVT_MENU_SELECT	When the menu button is selected, then EVT_MENU_SELECT is sent. After EVT_MENU_SELECT is received, then the menu bar is drawn on the screen for further selection.
EVT_PEN_DOWN	At the start, the menu object is in IDLE state. If the position of the pen is within the bounds, then EVT_MENU_ENTER is sent.
EVT_MENU_ENTER	The menu object is now in ENTER state. If the position of the pen while it is pen down is on one of the visible choice of menu object, then the current selected choice is highlighted
EVT_PEN_MOVE	If it is now in ENTER state, ➢ If one of the choices is being highlighted and the pen is moved within the bounds of the visible choices, then corresponding choice is highlighted. ➢ If one of the choices is being highlighted and the pen is moved out of the bounds of all visible choices, then EVT_MENU_EXIT is sent. If it is now in EXIT state, ➢ If the pen is moved back to one of the choices, then EVT_MENU_ENTER is re-sent again. ➢ If the pen is still out of the bounds of the menu bar, then nothing happens.
EVT_MENU_EXIT	The menu object is changed to EXIT state now. All the regions in the menu object are unhighlighted.
EVT_PEN_UP	If it is in ENTER state and the pen is lifted, ➢ If the highlighted region is one of the visible choice, then the choice becomes the selection

	of the menu object. EVT_MENU_SELECT_ITEM with the item number of the selected choice is sent.

7c. Data Structure

```
/* Structure: Menu_Attr */
struct Menu_Attr
{
        BOOLEAN             menu_drawn;
};
typedef struct Menu_Attr MenuAttr;

/* Structure: _Menu */
struct _Menu
{
        Identification      identification;
        ObjectBounds        bounds;
        BitmapTemplate      save_behind;
        USHORT              menu_num_items;
        USHORT              menu_max_num_items;
        SHORT               menu_selected_item;
        SHORT               menu_highlighted_item;
        BYTE **             menu_items;
        MenuAttr            menu_attr;
};
typedef struct _Menu Menu;
```

The following table shows all the parameters of the relative UI objects and discusses the function of them.

Parameters	*Function*
identification	Information to identification the object: • ui_object_id is the object ID of the object. • ui_object_type is the object type
bounds	The bounds field represents the Screen relative coordinates of the top left corner of the Menu. The bounds field contains the following parameters: xcoord Screen relative X-coordinate of the object.

	ycoord — Screen relative Y-coordinate of the object. width — Width of the object. height — Height of the object.
save_behind	Used to store the information of the Bitmap covered by the specified object. The structure contains the following parameters: xcoord — Screen relative X-coordinate of the top left corner of the covered image. ycoord — Screen relative Y-coordinate of the top left corner of the covered image. width — Width of the covered image. Height — Height of the covered image. compresed — Indicate whether the Bitmap is compressed or not. quantisation — Number of quantization levels per pixel. size — Size (No. of bytes) of the Bitmap. bitmap_data — Pointer to the locations of storing the covered image.
menu_num_items	Number of items in the menu.
menu_max_num_items	Maximum number of items displayed in the menu.
menu_selected_item	Item number of the selection.
menu_highlighted_item	Item number being highlighted.
menu_items	Pointer to an array of pointer to the text of the choices.
menu_attr	Pointer pointed to the MenuAttr structure, the structure of the MenuAttr contains the parameters menu_drawn menu_drawn — Indicate whether the Menu is drawn on screen or not.

7d. API Functions

The following API functions can be used to manipulate menu object.

- MenuCloseMenu

- MenuDeleteAllItems
- MenuDeleteItem
- MenuDeleteMenu
- MenuDrawMenu
- MenuGetAttrVisible
- MenuGetMenuItem
- MenuGetNumOfItems
- MenuGetTotalItems
- MenuInitMenu
- MenuInsertItem
- MenuSetAttrVisible
- MenuSetTotalItems

8. SCHEDULER LINE

8a. Characteristics and Behavior

Schline (Scheduler Line) is a particular object and provides the special feature for the Scheduler application. The schline can be divided into 4 types of regions. The first and second regions are the labels of the schline object, which displays the date and time setting on the pre-design area. The third region are combined with 21 bitmaps diagram, they are placed into 3 different columns to indicate the activity's status of the specified date. The fourth region are combined with 7 horizontal lines, they are placed into 7 different rows to indicate the status of the schedule.

8b. Event Flow

There are 7 events that are handled by the *SchlineHandleEvent*. They are:

- EVT_PEN_UP
- EVT_PEN_DOWN
- EVT_PEN_MOVE
- EVT_SCHLINE_ENTER
- EVT_SCHLINE_EXIT
- EVT_SCHLINE_SELECT

Among those events, EVT_SCHLINE_ENTER, EVT_SCHLINE_EXIT, EVT_SCHLINE_SELECT and EVT_SCHLINE_REPEAT are all generated by *SchlineHandleEvent*. The structures of the events can be found in the document of UI Events.

Events that are passed to SchlineHandleEvent	*Actions to be taken by SchlineHandleEvent*
EVT_PEN_DOWN	At starting, the schline object is in IDLE state. If the position of the pen is within the bounds, then EVT_SCHLINE_ENTER is sent.
EVT_SCHLINE_ENTER	The schline object is now in ENTER state. ➢ If the position of the pen is on one of the items in region 3, then the corresponding item is highlighted.
EVT_PEN_MOVE	If it is now in ENTER state, ➢ if the pen is moved out of the bounds of the schline object, then EVT_SCHLINE_EXIT is sent. ➢ if the pen is still within the bounds, then nothing happens. If it is now in EXIT state, ➢ if the pen is moved back to and within the bounds of the schline object, then EVT_SCHLINE_ENTER is sent. ➢ if the pen is still outside the bounds, then nothing happens.
EVT_SCHLINE_EXIT	The schline object is now in EXIT state. The display of the schline object is changed back to original one.
EVT_PEN_UP	If it is still ENTER state now, then EVT_SCHLINE_SELECT is sent and the display of the schline object is back to normal. But on the other hand, if it is EXIT state now, then state is changed back to IDLE state.

9c. Data Structure

```
/* Structure: Schline_Attr */
struct Schline_Attr
```

```
{
        BOOLEAN                 schline_drawn;
        BOOLEAN                 schline_visible;
        BOOLEAN                 schline_active;
        BOOLEAN                 schline_enable;
        BOOLEAN                 schline_enter;
};
typedef struct Schline_Attr SchlineAttr;

/* Structure: _Schline */
struct _Schline
{
        Identification          identification;
        ObjectBounds            bounds;
        ObjectBounds **         schline_string;
        ObjectBounds **         schline_line;
        ObjectBounds **         schline_bitmap0;
        ObjectBounds **         schline_bitmap1;
        ObjectBounds **         schline_bitmap2;
        BYTE *                  schline_bitmap_num;
        BYTE **                 schline_text;
        BYTE                    schline_text_highlight;
        SHORT                   schline_highlight_region;
        BitmapTemplate **       schline_bitmap_ptr;
        Boolean                 schline_mode;
        SchlineAttr             schline_attr;
};
typedef struct _Schline Schline;
```

The following table shows all the parameters of the relative UI objects and discusses the function of them.

Parameters	*Function*
identification	Information to identification the object: • ui_object_id is the object ID of the object. • ui_object_type is the object type
bounds	The bounds field represents the Screen relative coordinates of the top left corner of the Schline (Scheduler Line) Object. The bounds field contains the following parameters: xcoord Screen relative X-coordinate of the object. ycoord Screen relative Y-coordinate of the object. width Width of the object. height Height of the object.

schline_string	Store the location of the particular label (e.g. Mon 30, Tue 1)
schline_line	Store the location of 7 scheduler lines.
schline_bitmap0	Location of the all bitmaps in column 1.
schline_bitmap1	Location of the all bitmaps in column 2.
schline_bitmap2	Location of the all bitmaps in column 3.
schline_bitmap_ num	Bitmap number or No bitmap (0) in the corresponding area.
schline_text	Pointer reference to the address of the text.
schline_text_ highlight	To indicate the current date of the scheduler screen. The specify date will be highlight on the screen.
schline_highlight_ region	To indicate the highlight region of the specified line.
schline_bitmap_ptr	Pointer reference to the address of the particular bitmap diagram.
schline_mode	To indicate the Hours mode of the system.
schline_attr	Attribute of the Schline object. The schline_attr field contains the parameters schline_drawn, schline_active, schline_enable, schline_enter, schline_visible.

schline_drawn	Indicate whether the Scheduler Line is drawn on screen or not.
schline_visible	Indicate whether the Scheduler Line is visible on screen or not.
schline_active	Indicate whether the Scheduler Line is being used or not.
schline_enable	Indicate whether the Scheduler Line response to the pen action.
schline_enter	Indicate whether the Scheduler Line is entered by a pen or not.

8d. API Functions

The following API functions can be used to manipulate scheduler line object.

- SchlineDeleteSchline
- SchlineDrawSchline
- SchlineEraseSchline
- SchlineGetAttribute
- SchlineGetClickedRegion
- SchlineGetDateBitmaps
- SchlineGetDateText
- SchlineInitSchline
- SchlineSetAttribute
- SchlineSetHighlightText
- SchlineSetHourSettings
- SchlineSetLineBitmap
- SchlineSetLineLabel
- SchlineSetLineState

9. SCROLLBAR

9a. Characteristics and Behavior

Scrollbar is a support object and provides the scrolling feature for field and table object. The scrollbar can be divided into 3 different regions. The region is the scrollbar arrow. When user click on this region, the value of the scroll car (the colored box sit on the scrollbar) is increased or decreased by one line accordingly. The arrows of the scrollbar acts like repeat buttons. Therefore, the value of the scroll car is changing repeatedly while the arrows are held down continuously.

The second region is the region above and below the scroll car. When users click on it, the value of the scroll car is increased or decreased by the number of lines in one page.

The third region is the region of a scroll car itself. When user clicks on the scroll car and drags it to move along the scrollbar, then value of the scroll car changes continuously to reflect the current position of the line object and the table object.

The scrollbar would not update its display. Therefore, it is the job of the application to update the line object or the table object and the scrollbar after EVT_SCROLLBAR_REPEAT is received.

9b. Event Flow

There are 9 events that are handled by the *ScrollbarHandleEvent*. They are:

- EVT_PEN_UP
- EVT_PEN_DOWN
- EVT_PEN_MOVE
- EVT_SCROLLBAR_ENTER
- EVT_SCROLLBAR_REPEAT
- EVT_SCROLLBAR_EXIT
- EVT_SCROLLBAR_SELECT
- EVT_SCROLLBAR_ARROW_DELAY
- EVT_SCROLLBAR_ENTER_REPEAT

Among those events, *ScrollbarHandleEvent* sends EVT_SCROLLBAR_ENTER, EVT_SCROLLBAR_EXIT, EVT_SCROLLBAR_SELECT, EVT_SCROLLBAR_REPEAT, EVT_SCROLLBAR_ARROW_DELAY and EVT_SCROLLBAR_ENTER_REPEAT. EVT_SCROLLBAR_ARROW_DELAY and EVT_SCROLLBAR_ENTER_REPEAT are for the internal use of *ScrollbarHandleEvent* to generate delay time for clicking on the scrollbar. The structures of the events can be found in the document of UI Events.

Events that are passed to ScrollbarHandleEvent	***Actions to be taken by ScrollbarHandleEvent***
EVT_PEN_DOWN	At the start, the scrollbar object is in IDLE state. If the position of the pen is within the bounds, then EVT_SCROLLBAR_ENTER is sent.
EVT_SCROLLBAR_ENTER	The scrollbar object is now in ENTER state. ➢ If the position of the pen is on one of the arrow or on the scroll car, then the region is highlighted. ➢ If the position of the pen is on one of the arrow, then EVT_SCROLLBAR_REPEAT with the old and new value of the scroll car is sent.
EVT_PEN_MOVE	If it is now in ENTER state, ➢ If previous highlighted region is arrow and the position of the pen is now not in the bounds of the arrow region, then EVT_SCROLLBAR_EXIT is sent. ➢ If previous highlighted region is arrow and the position of the pen is still within the bounds of the arrow region, the new value is calculated and EVT_SCROLLBAR_REPEAT is sent.

	➢ If previous highlighted region is the scroll car and the position of the pen is within the bounds of the scrollbar, then the new value is calculated and EVT_SCROLLBAR_REPEAT is sent. ➢ If previous highlighted region is the scroll car and the position of the pen is out of the bounds of the scrollbar, then EVT_SCROLLBAR_EXIT is sent. If it is now in EXIT state, ➢ If the pen is moved back to the previous highlighted region, then new value is calculated and EVT_SCROLLBAR_REPEAT is re-sent again. ➢ If the pen is still out of the bounds of the previous highlighted region, then nothing happens.
EVT_SCROLLBAR_EXIT	The scrollbar object is changed to EXIT state now. All the regions in the scrollbar object are unhighlighted.
EVT_PEN_UP	If it is in ENTER state and the pen is lifted, all highlighted regions are unhighlighted and EVT_SCROLLBAR_SELECT with the current value of the scroll car is sent. If it is in the EXIT state, then the scrollbar object returns to IDLE state.
EVT_SCROLLBAR_REPEAT	EVT_SCROLLBAR_ARROW_DELAY or EVT_SCROLLBAR_ENTER_REPEAT (the sent-event depends on the clicked region) is sent to introduce a time delay between the first and second EVT_SCROLLBAR_REPEAT if the first EVT_SCROLLBAR_REPEAT is received. After a certain time delay, EVT_SCROLLBAR_REPEAT is sent periodically.

9c. Data Structure

```
struct Scrollbar_Attr
{
        BOOLEAN             scrollbar_drawn;
```

```
        BOOLEAN         scrollbar_active;
        BOOLEAN         scrollbar_visible;
        BOOLEAN         scrollbar_enable;
        BOOLEAN         scrollbar_enter;
        BOOLEAN         scrollbar_enter1;
        BOOLEAN         scrollbar_enter2;
};
typedef struct Scrollbar_Attr ScrollbarAttr;

struct _Scrollbar
{
        Identification      identification;
        ObjectBounds        bounds;
        USHORT              scrollbar_repeat_count;
        BitmapTemplate      scrollbar_arrow1;
        BitmapTemplate      scrollbar_arrow2;
        WORD                scrollbar_max;
        WORD                scrollbar_min;
        WORD                scrollbar_value;
        WORD                scrollbar_old_value;
        WORD                scrollbar_pagesize;
        WORD                scrollbar_draw_pagesize;
        WORD                scrollbar_total_num_lines;
        BYTE                scrollbar_type;
        BYTE                scrollbar_style;
        SHORT               scrollbar_save_pos_x;
        SHORT               scrollbar_save_pos_y;
        BYTE                scrollbar_clicked_region;
        BTYE  *             scrollbar_text;
        ScrollbarAttr       scrollbar_attr;
 };
typedef struct _Scrollbar Scrollbar;
```

The following table shows all the parameters of the relative UI objects and discusses the function of them.

Parameters	*Function*
identification	Information to identification the object: • ui_object_id is the object ID of the object. • ui_object_type is the object type
bounds	The bounds field represents the Screen relative coordinates of the top left corner of the Scrollbar. The bounds field contains the following parameters:

	xcoord	Screen relative X-coordinate of the object.
	ycoord	Screen relative Y-coordinate of the object.
	width	Width of the object.
	height	Height of the object.
scrollbar_repeat _count	Number of lines repeated displays in the field object.	
scrollbar_arrow1	Pointer pointed to the template structure of the scroll up arrow. The structure contains the following parameters:	
	xcoord	Screen relative X-coordinate of the scroll up arrow.
	ycoord	Screen relative Y-coordinate of the scroll down arrow.
	width	Width of the scroll up arrow.
	height	Height of the scroll down arrow.
	compresed	Indicate whether the Bitmap is compressed or not.
	quantisation	Number of quantization levels per pixel.
	size	Size (No. of bytes) of the scroll up arrow.
	bitmap_data	Pointer to the locations of storing the diagram of the scroll up arrow.
scrollbar_arrow2	Pointer pointed to the template structure of the scroll down arrow. The structure contains the following parameters:	
	xcoord	Screen relative X-coordinate of the top left corner of the scroll down arrow.
	ycoord	Screen relative Y-coordinate of the top left corner of the scroll down arrow.
	width	Width of the scroll down arrow.
	height	Height of the scroll down arrow.
	compresed	Indicate whether the Bitmap is compressed or not.
	quantisation	Number of quantization levels per pixel.
	size	Size (No. of bytes) of the scroll down arrow.
	bitmap_data	Pointer to the locations of storing the diagram of the scroll down arrow.
scrollbar_max	Position of the scroll car when the scrollbar is at the bottom. To compute this value, use the formula: Max value = Number of lines – Page size + Overlap value	
scrollbar_min	Position of the scroll car when the scrollbar is at the top. By default this value should be 0.	
scrollbar_value	Current value of the scrollbar.	

scrollbar_old_value	Previous value of the scrollbar.
scrollbar_pagesize	Number of lines to scroll when users scrolls one page.
scrollbar_draw_ pagesize	Draw purpose only.
scrollbar_total_ num_lines	Total number of lines in the scrollbar.
scrollbar_type	Indicate the type of the scrollbar. For example: SCROLLBAR_0 = Scrollbar with scroll arrow on both sides. SCROLLBAR_1 = Scrolled control bar type 1, used in "Voice Memo" application. (If the scroll car is shifted right, the region on the left hands side of the car will then be highlighted). SCROLLBAR_2 = Scrolled control bar type 2 (If the scroll car is shifted right, the region on the left hands side of the car will not be highlighted). SCROLLBAR_3 = Specify used in "System Setup" application.
scrollbar_style	Indicate is a horizontal or vertical bar. For example: VERTICAL or HORIZONTAL.
scrollbar_save_ pos_x	Save the window relative x-coordinate of the scrollbar.
scrollbar_save_ pos_y	Save the window relative y-coordinate of the scrollbar.
scrollbar_clicked_ region	Indicate the region clicked by the user. For example: SCROLL_NOT_HITTED ~ Not be clicked, SCROLL_UP_ARROW ~ To indicate on the upper arrow of the scrollbar. SCROLLBAR_UP_REGION ~ To indicate below the upper arrow and above the scroll car. SCROLLCAR_REGION ~ To indicate on the scroll car area. SCROLLBAR_DOWN_REGION ~ To indicate below the

		scroll car and above the lower arrow.
	SCROLL_DOWN_ARROW	~ to indicate on the lower arrow of the scrollbar.
scrollbar_text	Pointer to the text of the particular scrollbar. (Used by System Set up application)	
scrollbar_attr	Attribute of the Scrollbar object. The scrollbar_attr field contains the parameters scrollbar_drawn, scrollbar_active, scrollbar_enable, scrollbar_enter, scrollbar_enter1, scrollbar_enter2, scrollbar_visible.	
	scrollbar_drawn	Indicate whether the Scrollbar is drawn on screen or not.
	scrollbar_active	Indicate whether the Scrollbar is being used or not.
	scrollbar_enable	Indicate whether the Scrollbar response to the pen action.
	scrollbar_enter	Indicate whether the Scrollbar is entered by the pen or not.
	scrollbar_enter1	Indicate whether the scrollbar's up arrow is clicked or not.
	scrollbar_enter2	Indicate whether the scrollbar's down arrow is clicked or not.
	scrollbar_visible	Indicate whether the Scrollbar is visible on screen or not.

9d. API Functions

The following API functions can be used to manipulate scrollbar object.

- ScrollbarDeleteScrollbar
- ScrollbarDrawScrollbar
- ScrollbarEraseScrollbar
- ScrollbarGetScrollbar
- ScrollbarGetScrollbarText
- ScrollbarGetScrollbarVisible
- ScrollbarHardButtonSetScrollbar
- ScrollbarInitScrollbar
- ScrollbarSetScrollbar
- ScrollbarSetScrollbarDrawPagesize
- ScrollbarSetScrollbarText

- ScrollbarSetScrollbarType
- ScrollbarSetScrollbarVisible

10. STRING

10a. Characteristics and Behavior

String resource is used for displaying non-editable text of the screen. The object provides the ability for application to align the text on the string to left, center and right.

10b. Event Flow

String object is used for display purpose only. Therefore, there is response to any pen action on a string object.

10c. Data Structure

```
struct String_Attr
{
        BOOLEAN         string_drawn;
        BOOLEAN         string_visible;
};
typedef struct String_Attr StringAttr;

struct _String
{
        Identification  identification;
        ObjectBounds    bounds;
        ObjectBounds    screen_bounds;
        BYTE            string_color;
        BYTE            string_bg_color;
        BYTE            string_style;
        BYTE *          string_text;
        BYTE            text_alignment;
        BYTE            text_font;
        StringAttr      string_attr;
};
typedef struct _String String;
```

The following table shows all the parameters of the relative UI objects and discusses the function of them.

Parameters	*Function*
identification	Information to identification the object: • ui_object_id is the object ID of the object. • ui_object_type is the object type
bounds	If the object is pasted within a table, then the bounds field represents the **Table** relative coordinates of the top left corner of the String object. Otherwise represents the **Screen** relative coordinates of the top left corner of the String object. The bounds field contains the following parameters: xcoord Screen / Table relative X-coordinate of the object. ycoord Screen / Table relative Y-coordinate of the object. width Width of the object. height Height of the object.
screen_bounds	Screen relative coordinates of the top left corner of the String. The bounds field contains the following parameters: xcoord Screen relative X-coordinate of the object. ycoord Screen relative Y-coordinate of the object. width Width of the object. height Height of the object.
string_color	Color of the text. For example: COLOR_WHITE, COLOR_GREY1, COLOR_GREY2, COLOR_BLACK
string_bg_color	Background color of the string. For example: COLOR_WHITE, COLOR_GREY1, COLOR_GREY2, COLOR_BLACK
string_style	Style of the particular String. For example: STRING_STYLE_0 = Text only, no frame around the string object, STRING_STYLE_1 = With predefined frame around the string object. STRING_STYLE_2 = With predefined 3D frame around the string object. (e.g. the calculator display screen) STRING_STYLE_3= Particular style used on the application title. STRING_STYLE_4= Specify used in the “Anniversaries” application. STRING_STYLE_5 = Specify used in the “Scheduler” application.

string_text	Pointer to the text.	
text_alignment	Alignment of the text in the string. For example:	LEFT_ALIGN, CENTRE_ALIGN, RIGHT_ALIGN
text_font	Font of the text. For example:	SMALL_FONT, MEDIUM_FONT and LARGE_FONT
string_attr	Pointer pointed to the StringAttr structure, the structure of the StringAttr contains the parameters string_drawn and string_visible.	
	string_drawn	Indicate whether the String is drawn on screen or not.
	string_visible	Indicate whether the String is visible on screen or not.

10d. API Functions

The following API functions can be used to manipulate string object.

- StringDeleteString
- StringDrawString
- StringEraseString
- StringGetAttribute
- StringGetText
- StringInitString
- StringSetAttribute
- StringSetText

11. TABLE

10a. Characteristics and Behavior

Table object provides a best way for the application to organize the layout of the application properly. Application can have more spaces virtually by placing UI objects or

text in cells of a table object and by using scrollbar to scroll the objects up or down. This feature gives flexibility to the whole PDA developing environment.

There are different types of things that can be placed in a cell of a table object. They are text, value, and UI objects. Cells can be used to store text or value. Both text and value are displayed in a cell as a text, but the value can be used for calculation in some applications. In addition, different UI objects can be placed in cells of table, but the size of the UI object must be smaller than the size of the table object.

There are two selecting modes when user clicks on the cell that stores text or value. One mode of selection is that the text in the cell will be unhighlighted back after the pen is lifted. The other mode of selection is that the text in the cell will stay highlighted after the pen is lifted from the cell. The second mode of selection provides the effect of selecting and deselecting. It is quite useful for some application.

By default, table resources contain three different types: 0 = No line between the bounds, 1 = with vertical lines in the table and 2 = with vertical and horizontal lines in the table. User can base on the requirement of the applications to select the type.

11b. Event Flow

There are 6 events that are handled by the *TableHandleEvent*. They are:

- EVT_PEN_UP
- EVT_PEN_DOWN
- EVT_PEN_MOVE
- EVT_TABLE_ENTER
- EVT_TABLE_EXIT
- EVT_TABLE_SELECT

Among those events, EVT_TABLE_ENTER, EVT_TABLE_EXIT and EVT_TABLE_SELECT are all generated by *TableHandleEvent*. The structures of the events can be found in the document of UI Events.

Events that are passed to TableHandleEvent	*Actions to be taken by TableHandleEvent*
EVT_PEN_DOWN	At the start, the table object is in IDLE state. If the position of the pen is within the bounds of one of the cell that contains text or value, then EVT_TABLE_ENTER is sent. Otherwise, nothing happens.
EVT_TABLE_ENTER	The table object is now in ENTER state. The cell the pen is on is highlighted.

EVT_PEN_MOVE	If it is now in ENTER state, ➢ If the pen is out of the bounds of the cell that was highlighted, then EVT_TABLE_EXIT is sent. ➢ If the pen is still within the bounds of the cell that was highlighted, then nothing happens. If it is now in EXIT state, ➢ If the pen is moved back to the previous highlighted cell, then EVT_TABLE_ENTER is re-sent again. ➢ If the pen is still out of the bounds of the previous highlighted cell, then nothing happens.
EVT_TABLE_EXIT	The table object is changed to EXIT state now. The cell that was highlighted in the table object is unhighlighted.
EVT_PEN_UP	If it is in ENTER state and the pen is lifted, the cell is highlighted or unhighlighted depending on the mode of selection of the table object. The table object returns to IDLE mode. If it is in the EXIT state, then the table object returns to IDLE state.

11c. Data Structure

```
struct Table_Attr
{
        BOOLEAN         table_drawn;
        BOOLEAN         table_scrollbar;
        BOOLEAN         table_enable;
        BOOLEAN         table_active;
        BOOLEAN         table_enter;
        BOOLEAN         table_visible;
        BOOLEAN         table_highlight_enable;
};
typedef struct Table_Attr TableAttr;

/* Structure: Table_Items */
struct Table_Items
{
```

```
        BYTE            table_data_type;
        ObjectID        table_item_ui_id;
        BYTE            table_font;
        WORD            table_value;
        BYTE            table_display_alignment;
        BYTE            table_text_color;
        BYTE            table_text_bg_color;
        BOOLEAN         table_cell_highlight;
        BOOLEAN         table_cell_has_bitmap;
        BYTE *          table_text;
};
typedef struct Table_Items TableItems;

/* Structure: _Table */
struct _Table
{
        Identification  identification;
        ObjectBounds    bounds;
        USHORT          table_num_column;
        USHORT          table_num_row;
        USHORT          table_current_row;
        USHORT          table_current_col;
        TableItems **   table_item_ptr;
        SHORT *         table_column_width;
        SHORT *         table_row_height;
        USHORT          table_num_col_display;
        USHORT          table_num_row_display;
        USHORT          table_top_row_num;
        USHORT          table_left_col_num;
        BYTE            table_style;
        BYTE            table_bg_color;
        BitmapTemplate  table_cell_bitmap;
        TableAttr       table_attr;
};
typedef struct _Table Table;
```

The following table shows all the parameters of the relative UI objects and discusses the function of them.

Parameters	*Function*
identification	Information to identification the object: • ui_object_id is the object ID of the object. • ui_object_type is the object type
bounds	The bounds field represents the table relative coordinates of the

	top left corner of the Table. The bounds field contains the following parameters: xcoord Screen relative X-coordinate of the object. ycoord Screen relative Y-coordinate of the object. width Width of the object. height Height of the object.
table_num_column	Total number of columns in the table.
table_num_row	Total number of row in the table.
table_current_row	Row of the table set to current. (Row number start from 0)
table_current_col	Column of the table set to current. (Column number start from 0)
table_item_ptr	Pointer to an array of the Table_Items structures. This structure contains the following parameters: table_data_type: Data type of the item. (e.g. 0 = Text, 1 = Value / Number or 2 = UI Object) table_item_ui_id: If the item is UI object, then the object ID of the object is placed in here, Otherwise, Null is placed instead table_font: Font size of the text in the table object. table_value: Value / number of the item. table_display_alignment: Alignment of the text in the table. For example: LEFT_ALIGN, CENTRE_ALIGN, RIGHT_ALIGN table_text_color: Color of the text. For example: COLOR_WHITE, COLOR_GREY1, COLOR_GREY2, COLOR_BLACK table_text_bg_color: Background color of the table. For example: COLOR_WHITE, COLOR_GREY1, COLOR_GREY2, COLOR_BLACK table_cell_highlight: Indicate the specified cell is highlighted. table_cell_has_bitmap: Indicate the specified cell has a bitmap placed on it. table_text: Pointer to the text.
table_column_width	Pointer to a list of column width.

table_row_ height	Pointer to a list of row height.
table_num_col_ display	Number of column on display.
table_num_row_ display	Number of row on display.
table_top_row_ num	Row number of the top row that is on display.
table_left_col_ num	Column number of the left column that is on display.
table_style	Style of the particular Table. For example: TABLE_STYLE_0 ~ No line in the bounds, TABLE_STYLE_1 ~ with vertical lines in the table, TABLE_STYLE_2 ~ with vertical and horizontal lines in the table
table_bg_color	Background color of the table. For example: COLOR_WHITE, COLOR_GREY1, COLOR_GREY2, COLOR_BLACK
table_cell_bitmap	Pointer pointed to the template structure of the Bitmap diagram in the specified cell. The structure contains the following parameters: xcoord — Screen relative X-coordinate of the Bitmap. ycoord — Screen relative Y-coordinate of the Bitmap. width — Width of the Bitmap. Height — Height of the Bitmap. compresed — Indicate whether the Bitmap is compressed or not. quantisation — Number of quantization levels per pixel. size — Size (No. of bytes) of the Bitmap. bitmap_data — Pointer to the locations of storing the diagram.
table_attr	Pointer pointed to the TableAttr structure, the structure of the TableAttr contains the parameters table_drawn, table_scrollbar, table_enable, table_active, table_enter, table_visible, table_highlight_enable. table_drawn — Indicate whether the Table is drawn on screen or not.

table_scrollbar	Indicate whether the table contains a scrollbar or not.
table_enable	Indicate whether the table is already enabled on the display or not.
table_active	Indicate whether the table is active on the display or not.
table_enter	Indicate whether the specified cell is entered by a pen or not.
table_visible	Indicate whether the Table is visible on screen or not.
table_highlight enable	Indicate the text / data in the cell is highlighted or not. When the pen is clicked on the corresponding region.

11d. API Functions

The following API functions can be used to manipulate table object.

- TableCheckCellHasBitmap
- TableCheckCellHighlight
- TableCheckHighlightEnable
- TableDeleteTable
- TableDrawTable
- TableEnableTable
- TableEraseTable
- TableGetAttributes
- TableGetCellBounds
- TableGetClickedCell
- TableGetColumnWidth
- TableGetItemText
- TableGetItemType
- TableGetItemValue
- TableGetNumOfColumns
- TableGetNumOfRows
- TableGetNumOfRowsDisplayed
- TableGetRowColOfSelection
- TableGetRowHeight
- TableGetTableBounds
- TableGetTopRowNum
- TableInitTable
- TableSetAttributes
- TableSetBounds
- TableSetCellHasBitmap
- TableSetColumnWidth

- TableSetHighlightCell
- TableSetHighlightEnable
- TableSetItemText
- TableSetItemType
- TableSetItemValue
- TableSetRowHeight
- TableSetTopRowNum
- TableUpdateNumRowDisplay
- TableUpdateObjectScreenBounds
- TableUpdateTable

12. TEXTBOX

12a. Characteristics and Behavior

Textbox resource provides the ability for the application to input editable text. The text, which is in the textbox object, can only be displayed with one single line. It is the main difference between field object and text object. There are also many features that are implemented and supported by the textbox object.

They are:

- Drag-scrolling and Drag-selection
- CUT, PASTE and COPY
- Special keys – HOME, END and BACKSPACE
- Indication of insertion point
- Switching between viewing mode and editing mode
- Protection by setting maximum number of characters

It is the same as field object. In order to give flexibility to application to filter the characters to a textbox object, the routine for adding key-in character to the string of textbox object is separated from the *TextboxHandleEvent.* Therefore, the EVT_KEY with the visible character is sent to application layer for further process and the *TextboxHandleEvent* would not handle EVT_KEY with visible character.

12b. Event Flow

There are 6 events that are handled by the *TextboxHandleEvent*. They are:

- EVT_PEN_UP
- EVT_PEN_DOWN
- EVT_PEN_MOVE

- EVT_TEXTBOX_ENTER
- EVT_TEXTBOX_SELECT
- EVT_KEY
- EVT_TEXTBOX_CHANGED
- EVT_TEXTBOX_MODIFIED
- EVT_TEXTBOX_JOT_PASTE_STRING

Among those events, *TextboxHandleEvent* sends all EVT_TEXTBOX_ENTER, EVT_TEXTBOX_MODIFIED, EVT_TEXTBOX_CHANGED and EVT_TEXTBOX_SELECT. *JotHandleEvent* sends EVT_TEXTBOX_JOT_PASTE_STRING to request pasting symbol onto textbox object. The structures of the events can be found in the document of UI Events.

Events that are passed to TextboxHandleEvent	***Actions to be taken by TextboxHandleEvent***
EVT_PEN_DOWN	At the start, the textbox object is in IDLE state. If the position of the pen is within the bounds, then EVT_TEXTBOX_ENTER is sent.
EVT_TEXTBOX_ENTER	The textbox object is now in ENTER state. The insertion point of the textbox object is set to display on the position of the pen.
EVT_PEN_MOVE	If it in now in ENTER state, ➢ when the pen moves around, the text between the position of the first pen down and the current position of the pen is highlighted. After highlighting, the insertion point disappears. ➢ when the pen moves to the left or right of the textbox, the text in the textbox will scroll to left or right accordingly.
EVT_PEN_UP	If it is in ENTER state, then when the pen is lifted, EVT_TEXTBOX_SELECT is sent. The textbox object is back to IDLE state and the highlighted section of text is still on the textbox object.
EVT_KEY with BACKSPACE	The character on the left of the insertion point is deleted. EVT_TEXTBOX_CHANGED is sent.
EVT_KEY with CUT	The text that is being highlighted is cut to clipboard. EVT_TEXTBOX_CHANGED is sent.
EVT_KEY with COPY	The text that is being highlighted is copied to clipboard.

EVT_KEY with PASTE	The text in clipboard is pasted to the textbox object at the insertion point or to replace the highlighted section in the textbox object. EVT_TEXTBOX_CHANGED is sent.
EVT_KEY with HOME	The first line of text in the string of the textbox object is displayed at the top of the textbox object and the insertion point is positioned in the top-left corner of the textbox object.
EVT_KEY with END	The last line of text in the string of the textbox object is displayed at the bottom of the textbox object and the insertion point is positioned in the bottom-right corner of the textbox object.
EVT_KEY with LEFT ARROW	The insertion point is moved to the left for one character position. If the current position of the insertion point is already in the left margin, then the insertion point will be moved to the end of the previous line.
EVT_KEY with RIGHT ARROW	The insertion point is moved to the right for one character position. If the current position of the insertion point is already in the right margin, then the insertion point will be moved to the start of next line.

12c. Data Structure

```
/* Structure: Textbox_Attr */
struct Textbox_Attr
{
        BOOLEAN         textbox_drawn;
        BOOLEAN         textbox_active;
        BOOLEAN         textbox_enable;
        BOOLEAN         textbox_dirty;
        BOOLEAN         textbox_highlight;
        BOOLEAN         textbox_insert_pt_visible;
        BOOLEAN         textbox_visible;
};
typedef struct Textbox_Attr TextboxAttr;

/* Structure: _Textbox */
struct _Textbox
{
```

```
    Identification      identification;
    ObjectBounds        bounds;
    ObjectBounds        screen_bounds;
    BYTE *              textbox_string;
    BYTE                textbox_style;
    BYTE                textbox_back_line;
    BYTE                textbox_font_id;
    BYTE                textbox_font_color;
    BYTE                textbox_background_color;
    WORD                textbox_max_chars;
    WORD                textbox_current_num_chars;
    WORD                textbox_left_char_pos;
    WORD                textbox_right_char_pos;
    WORD                textbox_num_chars_displayed;
    SHORT               textbox_insert_pt_x;
    SHORT               textbox_insert_pt_y;
    WORD                textbox_insert_pt_char_pos;
    BYTE                textbox_insert_pt_movement;
    WORD                textbox_highlight_start_char;
    WORD                textbox_highlight_end_char;
    WORD                textbox_highlight_length;
    USHORT              textbox_repeat_count;
    TextboxAttr         textbox_attr;
};
typedef struct _Textbox Textbox;
```

The following table shows all the parameters of the relative UI objects and discusses the function of them.

Parameters	*Function*
identification	Information to identification the object: • ui_object_id is the object ID of the object. • ui_object_type is the object type
bounds	If the object is pasted within a table, then the bounds textbox represents the **table** relative coordinates of the top left corner of the Textbox. Otherwise represents the **Screen** relative coordinates of the top left corner of the Textbox. The bounds textbox contains the following parameters: xcoord Screen / Table relative X-coordinate of the object. ycoord Screen / Table relative Y-coordinate of the object. width Width of the object.

	height Height of the object.
screen_bounds	Screen relative coordinates of the top left corner of the Textbox. The bounds textbox contains the following parameters: xcoord Screen relative X-coordinate of the object. ycoord Screen relative Y-coordinate of the object. width Width of the object. Height Height of the object.
textbox_string	Pointer pointed to the text string in the textbox object.
textbox_style	Style of the particular Textbox. For example: TEXTBOX_STYLE_0 = No frame around the textbox object. TEXTBOX_STYLE_1 = With frame around the textbox object. TEXTBOX_STYLE_2 = With 3D frame around the text object.
textbox_back_line	Style of the lines under the textbox's text. For example: NO_LINE = Without lines under the text. DOT_LINE = With dot lines. GREY_LINE = With gray lines. (The color of the lines is predefined.)
textbox_font_id	Font type of the textbox's text. For example: SMALL_FONT, MEDIUM_FONT and LARGE_FONT.
textbox_font_color	Color of the textbox's text. For example: COLOR_WHITE, COLOR_GREY1, COLOR_GREY2 and COLOR_BLACK
textbox_background_color	Background color of the textbox. For example: COLOR_WHITE, COLOR_GREY1, COLOR_GREY2 and COLOR_BLACK
textbox_text_alignment	Alignment of the text in the textbox. For example: LEFT_ALIGN, CENTRE_ALIGN, RIGHT_ALIGN
textbox_max_chars	Maximum number of characters in the textbox object.
textbox_current_num	Current number of characters in the string displayed by the

_chars	textbox object; the null-terminator is excluded.
textbox_left_char_pos	The character position of the leftmost displaying character.
textbox_right_char_pos	The character position of the rightmost displaying character.
textbox_num_chars_displayed	The current number of characters that are being displayed
textbox_insert_pt_x	Column position of the insertion point.
textbox_insert_pt_y	Row position of the insertion point.
textbox_insert_pt_char_pos	The character position beside the insert point.
textbox_insert_pt_movement	Movement direction in the textbox. For example: NO_MOVEMENT, MOVE_RIGHT MOVE_LEFT
textbox_highlight_start_char	Starting character position of the current selection.
textbox_highlight_end_char	Ending character position of the current selection.
textbox_highlight_length	Length of the current selection. If the *textbox_highlight_start_char* is equal to the *textbox_highlight_end_char*, there is no selection.
textbox_repeat_count	Number of lines repeated displays in the textbox object.
textbox_attr	Attribute of the Textbox object. The textbox_attr textbox contains the parameter textbox_drawn, textbox_active, textbox_enable, textbox_dirty, textbox_highlight, textbox_insert_pt_visible and textbox_visible. textbox_drawn — Indicate whether the Textbox is drawn on screen or not. textbox_active — Indicate whether the Textbox is being used or not. textbox_enable — Indicate whether the Textbox response to the pen action. textbox_dirty — Indicate the textbox object has been changed. (e.g. cut, paste) textbox_highlight — Indicate whether the textbox is highlighted or not.

	textbox_insert_pt_visible	Indicate whether the insert point of the textbox is visible or not.
	textbox_visible	Indicate whether the Textbox is visible on screen or not.

12d. API Functions

The following API functions can be used to manipulate textbox object.

- TextboxAddKeyInChar
- TextboxDeleteString
- TextboxDeleteTextbox
- TextboxDirty
- TextboxDrawTextbox
- TextboxEraseTextbox
- TextboxGetAttribute
- TextboxGetCurrentHighlightedSelection
- TextboxGetFont
- TextboxGetInsertPointPosition
- TextboxGetLeftCharPos
- TextboxGetMaxNumChars
- TextboxGetNumOfChars
- TextboxGetNumOfCharsDisplayed
- TextboxGetRightCharPos
- TextboxGetTextPointer
- TextboxGetTextboxBounds
- TextboxInitTextbox
- TextboxInsertString
- TextboxPasteString
- TextboxSetAttribute
- TextboxSetBounds
- TextboxSetDirty
- TextboxSetFont
- TextboxSetHighlightSelection
- TextboxSetInsertPointOff
- TextboxSetInsertPointOn
- TextboxSetInsertPointPositionByCharPos
- TextboxSetInsertPointPositionByXY
- TextboxSetLeftCharPos
- TextboxSetMaxNumChars
- TextboxSetRightCharPos
- TextboxSetText

- TextboxUndo