Micro Windows
(UWindows)

I. Introduction:

UWindows is a Graphical User Interface (GUI) library and frame work built upon Circle_OS. UWindows enable programmers to create effective user interfaces in few simple and easy steps. This reliefs programmers from the tedious work of creating GUI and handling LCDs and allows them to concentrate more on the application rather than the GUI. UWindows currently support forms, labels, buttons, edit boxes, check boxes and icons. For every UWindow object a user can modify the list of properties of that object to determine how and where an object should appear. Also each object has list of events (Touch Screen Click, Joy Stick Move, Timer, etc..) which can be attached to it. By utilizing these events, a program is no more flow driven but event driven. Programmers can add multiple applications to UWindows main windows. All applications added to UWindows will be available to the user to pick from them just like any desktop. UWindows also provides a list of utilities for which a user can make use of mainly including memory utilization functions (dynamic allocation and freeing, memory copy and memory set) and Touch Screen calibration. Finally, UWindows is designed to be configurable with its features enabled or disabled based on the size and utilities requested by the user. UWindows opens a room for a numerous number of applications. Included as a sample with the library: Calculator, Clock, Phone Book and a simple Hello World example used for demo.

II. Uwindow Objects:

UWindows is based on objects. Every new item is considered a new Uwindow object. The behavior of the object and its shape changes based on the properties assigned to that object. All objects have a common list properties that they share together ex: x and y positions, length, height, caption, etc. Extra properties can be assigned to an object based on the type of the object. For example a list box should have List Box Properties object assigned to its extra properties (extra to the common properties) to determine in more specific details how the list box should behave.

Except for the main UWindows object (UWindows) every other object must have a parent. An object have a single parent but can have multiple sons. The arrangement of siblings and sons determine how the object should appear in user interface when rendered.

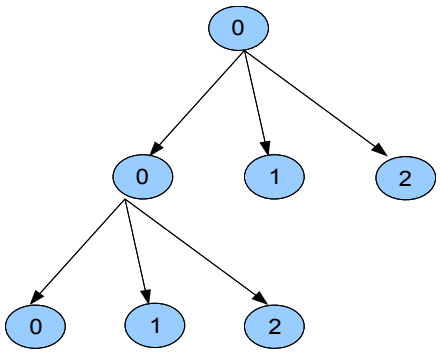

Figure 1 Structure of Uwindow Objects

As shown in figure 1, the UWindows objects constitutes a tree. When rendering the image this tree is evaluated and all objects that is visible and should be rendered are added to the image. As the figure shows, the objects in the tree are visited in the right most first order. First the object is rendered then the first son and so on. So for the figure shown object 0 is the first rendered object followed by object 0.0, 0.0.0, 0.0.1, 0.0.2 and 0.1. The order of adding items makes significance when two or more visible items overlap in a region. The order of addition determine which object is in background and which is in foreground (As shown in figure 2).

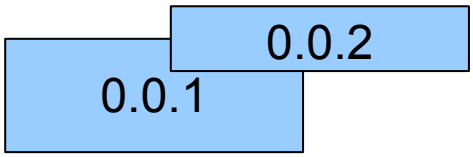

Figure 2 Layout out of two overlapping objects

III. Character Set:

Each object type supported by Uwindows has a default way of operation to communicate with the user. For example a check box is usually used to indicate a boolean value to the user. A click on the check box will toggle the value of the check box between 1 and 0. Edit Box is another way of retrieving date from users. Edit boxes get its caption by writing to it. Writing to an LCD can be a tedious job. For this, UWindows has created a character set map. Character Set Map is a map of all the alphabetic characters. It will pop up every time an Edit box is clicked This edit box will be known as the active edit box. The output of the character map will be applied to this edit box. The user can cancel the map and continue to work normally or can use the map to edit the caption of the edit box which can later convey to some user data (ex: Name, Telephone No., etc..).

Figure 3 show the design of the map

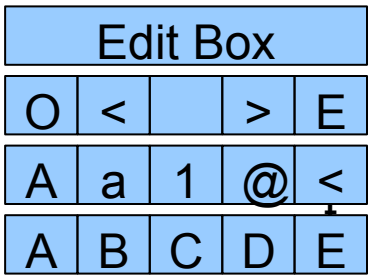

Figure 3 Character Set Map

The Items in the character set map include:

- Edit Box: This is an edit box that shows the current value of the caption. Pressing OK at any time will map this value to the actual active edit box.
- OK Button: Confirm the operation and map the caption to the active edit box
- Ext Button: Cancel operation and return back to normal form. Nothing will be done to the active edit box. It will keep its last value.
- A,a,1,@,<- Buttons: These are the group buttons. Every button represents a group of characters as listed here:
    - A: Group of upper case Alphabetics: A,B,C,D,E,etc.....
    - a: Group of lower case Alphabetics: a,b,c,d,e,etc....
    - 1: Group of numbers: 1,2,3,4,5,etc...
    - @: Group of misc. characters: +,-,*,.,=,@,%, etc...
    - ← This is back space to delete a character

Pressing on a group button will display the first 5 characters available for this group in the last row. For example when pressing A group button, the last row of the Map would show A,B,C,D,E. If the user later decided to enter numbers he will need to press the 1 group button. On doing this the last line would show 1,2,3,4,5.

- < , > Scroll Buttons, These are used to show the next or previous 5 characters in a group. To continue our last example, if the user clicked the > button, the last row would show 6,7,8,9,0
- Last Row Characters: This is the actual characters pressing on any character would add it to the caption of the edit box. In case of error user can use back space button (←) to delete the erroneous character
- SPC: this is the character between the two Scrol buttons. This character is also available in the misc character group, however because of its wide usage we have added a short cut in the main map for ease of use.

At any time the user can apply the changing and return by pressing OK or cancel the whole operation and return back by pressing Ext button.

IV. Application Program Interface (API):

In this section we discuss the group of properties and APIs needed to control the behavior and shape of an object.

The following type structure is used to represent any UWindows object:

```
typedef struct
{
    struct  list_head   node;
    struct  list_head   items;
    UW_Type             type;
    u8                  x,y;
    s32                 Absx,Absy;
    s32                 length,height;
    bool                visible,enable;
    u8  *               caption;
    u8                  captionlen;
    s32                 value;
    UW_CallBack         CallBacks;
    void *              ExtraProperties;
} UW_Window;
```

some of the items of this structure has to do with the internal working of the Uwindow library, while the rest are common properties which adjust the behavior of the object. A brief summary of these properties is listed here:

Name:	type
Type :	UW_Type
Valid values:	UW_FORM,UW_BUTTON,UW_LABEL,UW_EDIT,UW_LISTBOX,UW_CHECKBOX, UW_ICON.
Descriptions:	This variable is the main controller of the behavior of the object. It determines weather the object is a form, a button, a list box or any thing else.

Name:	x,y
Type :	signed integer
Valid values:	from 0 to 128. values other than this will not be drawn. However if any object has started in an invalid position yet some parts of it extened to come to the valid region, the valid part of the object is rendered in the image.
Descriptions:	The staring X and Y positions of the object. Valid values are within the screen

Name:	length and height
Type :	signed integer
Valid values:	from 0 to 128.
Descriptions:	Object width and height

Name:	Caption
Type :	Unsigned char *
Valid values:	All
Descriptions:	The caption is the string displayed on the object. Based on the object type the location of the caption will change. Note caption is only a pointer so there is no memory attached with this property it is the job of the programmer to secure the memory location for the caption. If a maximum size of the caption is known, the allocation can be done statically. In case of a varying caption (like in edit boxes where user enter a variable length of letters) user can use the dynamic allocation tool provided by UWindows.

Name:	captionlen
Type :	unsigned char
Valid values:	from 0 to 255.
Descriptions:	Length of the caption

Name:	Extraproperties
Type :	void *
Valid values:	NULL, Icon_Image pointer, UW_ListBox pointer.
Descriptions:	a pointer to the extra properties object of the UWindow object. Based on the type of the object the extra property field should casted as shown in the following table:

UW_FORM	:	NULL
UW_LABEL	:	NULL
UW_EDIT	:	NULL
UW_BUTTON	:	NULL
UW_ICON	:	u8 * (pointer to icon image)

UW_LISTBOX	:	UW_ListBox *
UW_CHECKBOX:		u8 * (Value of check box)

PS: an icon image is a buffer of [4 * 32] bytes to map a 32 X 32 Bit image. Each bit in the buffer map to a singe pixel either Black or while. The first 4 bytes map to the first column of the Icon Image and so on.

The programmer can directly access these variables, However it is recommended that the user should not attempt to changes these properties manually but through the specific API for these properties. The list of APIs provided by UWindows are:

1. UW_SetType:

ProtoType:

UW_SetType(UW_Window * pWindow,UW_Type Type)

Parameters:

pWindow	:	Pointer to UWindow object
Type	:	Type of Window

Description and Usage:

Set the type of the UW_Window to the required type. This function is rarely used. The type is set initally during add process and needn't be changed later.

2. UW_SetPosition:

ProtoType:

UW_SetPosition(UW_Window * pWindow,s32 x,s32 y)

Parameter:

pWindow	:	Pointer to UWindow object
x	:	x position of Window
y	:	y position of Window

Description and Usage:

Set the coordinates of the UW_Window object. The default values is 0,0.

3. UW_SetSize:

ProtoType:

UW_SetSize(UW_Window * pWindow,u8 length,u8 height)

Parameter:

pWindow	:	Pointer to UWindow object
x	:	length of Window
y	:	height of Window

Description and Usage:
Set the size of the UW_Window object. The default values is 20,20.

4. UW_SetPosition:

ProtoType:
UW_SetCaption(UW_Window * pWindow,u8 * pcatpion,u8 cpationlen)
Parameter:

pWindow	:	Pointer to UWindow object
pcaption	:	pointer to char array
captionlen	:	length of array

Description and Usage:
Set the caption of the UW_Window object. The default values is NULL and length is 0.

5. UW_SetVisible:

ProtoType:
UW_SetVisible(UW_Window * pWindow, bool visible)
Parameter:

pWindow	:	Pointer to UWindow object
visible	:	visibility

Description and Usage:
Set the visibilty of the UW_Window object. The default values is 1 (visible).

6. UW_SetEnable:

ProtoType:
UW_SetEnable(UW_Window * pWindow, bool Enable)
Parameter:

pWindow	:	Pointer to UWindow object
Enabled	:	Enablity

Description and Usage:
Set UW_Window object is enabled. The default values is 1 (Enabled).

7. UW_SetOnClick:

ProtoType:

UW_SetOnClick(UW_Window * pWindow,(void (*OnClick)(UW_Window * pWindow) ptrFunc)

Parameter:

pWindow : Pointer to UWindow object
ptrFunc : pointer to click callback function

Description and Usage:

Set the click callback of the UW_Window object. The default values is NULL.

PS: pWindow is provided with the call back to indicate the object being clicked. This allows the user to assign multiple objects to the same call back and use this parameter to handle each object differently

8. UW_SetOnButtonPress:

ProtoType:

UW_SetOnButtonPress(UW_Window * pWindow,(void (*OnButtonPress) (UW_Window * pWindow) ptrFunc)

Parameter:

pWindow : Pointer to UWindow object
ptrFunc : pointer to Button Press callback function

Description and Usage:

9. UW_SetOnTimer:

ProtoType:

UW_SetOnTimer(UW_Window * pWindow,(void (*OnTimer)(UW_Window * pWindow) ptrFunc, u32 Timer)

Parameter:

pWindow : Pointer to UWindow object
ptrFunc : pointer to Timer callback function
Timer : Duration of Time out in milliseconds

Description and Usage:

Set the Timer callback of the UW_Window object. The default values is NULL.

10. UW_SetValue:

ProtoType:

UW_SetValue(UW_Window * pWindow, s32 Value)

Parameter:

pWindow : Pointer to UWindow object

Value : Value of the Object

Description and Usage:

Set value of UW_Window object . The default values is 0. This value is mainly used with check boxes but user can use it as any value indicator for the object.

11. UW_ListBoxAddItem

ProtoType:

void UW_ListBoxAddItem(UW_Window *ListBox, UW_Window *Item)

Parameter:

pWindow : Pointer to UWindow object treated as List Box

Item : A list box item. This is a UWindow object treated as Edit box

Description and Usage:

This function will add a new Item to the list box. The function only adds an item to list it doesn't create the item. It is up to the user to create the new item.

12. UW_ListBoxRemoveItem

ProtoType:

void UW_ListBoxRemoveItem(UW_Window *ListBox, UW_Window *Item)

Parameter:

pWindow : Pointer to UWindow object treated as List Box

Item : A list box item. This is a UWindow object treated as Edit box

Description and Usage:

This function will remove Item from the list box.

13. UW_ListBoxGetActiveItem

ProtoType:

void UW_ListBoxGetActiveItem(UW_Window *ListBox)

Parameter:

pWindow : Pointer to UWindow object treated as List Box

Return:

A pointer to the active list box item.

Description and Usage:
This function will return a pointer to the active list box Item.

14.UW_ListBoxGetItem

ProtoType:
void UW_ListBoxGetItem(UW_Window *ListBox, u32 Index)
Parameter:
pWindow : Pointer to UWindow object treated as List Box
index : index of the Item

Return:
A pointer to the item with index index.

Description and Usage:
This function will return a pointer to the list box Item with index index.

15.UW_ListBoxGetIndex

ProtoType:
void UW_ListBoxGetIndex(UW_Window *ListBox, UW_Window *Item)
Parameter:
pWindow : Pointer to UWindow object treated as List Box
Item : A list box item. This is a UWindow object treated as Edit box

Description and Usage:
This function will return the index of the list box item and -1 if not found.

16.UW_Malloc

ProtoType:
void * UW_Malloc(unsigned long nbyes)
Parameter:
nbytes : Number of bytes required

Return:
pointer to allocated memory block.

Description and Usage:
This function will allocate nbytes of memory from the virtual heap reserved by UWindows.

17.UW_Free

ProtoType:

void * UW_Free(void * ptr)

Parameter:

ptr : pointer to allocated heap

Description and Usage:

This function will free allocated memory from the virtual heap pointer to by ptr.

18.UW_MemCpy

ProtoType:

void * UW_MemCpy(void *s1, const void *s2, int n)

Parameter:

s1 : pointer to destination
s2 : pointer to source
n : number of bytes to be copied

Description and Usage:

This function will copy n bytes from source to destination

19.UW_MemSet

ProtoType:

void * UW_Set(void *s1, unsigned char val, int n)

Parameter:

s1 : pointer to destination
val : value to be set
n : number of bytes to be set

Description and Usage:

This function will set n bytes pointed to by s1 to the value val.

V. Hello UWindows!:

In This section we will demonstrate how to use UWindows to create a simple effective user interface. The objective is to create an application which displays the words “Hello UWindows!”. To make it more interesting we will add an extra button when pressed will toggle the visibility of the “Hello UWindows!” message.

First, Let's declare all the UWindows objects needed for the application. For this demo we need 4 UWindows objects:

- Icon: will be added to the main window. When clicked the Icon would show the Hello Form.
- Form: which will hold the Message and the button
- Label: to hold the required message
- Button: to toggle the visibility of the label.

The following code declares the 4 UWindows objects:

```
#include "UWindows.h"

static UW_Window  HelloIcon;
static UW_Window  HelloForm;
static UW_Window  HelloBtn,HelloLabel;
```

To display captions we need a list of constant strings, defined as:

```
static const char IconCaption[]="Helo";
static char LabelCaption[] = "Hello UWindows!";
static char BtnCaption[]= "Show/Hide";
```

This application requires 4 functions

I. Show Form:

```
/* Handler of Icon Click to show Form    */
void HelloEnter(UW_Window * pWindow)
{
   UW_SetVisible(&HelloForm,1);
}
```

II. Hide Form:

```
/* Handler of Form Click to Hide Form    */
void HelloExit(UW_Window * pWindow)
{
   UW_SetVisible(&HelloForm,0);
```

```
}
```

III. Toggle Lable Visibilty:

```
/* Handler of Btn Click to toggle vision of Label
      */
void HelloBtnClick(UW_Window * PWindow)
{
  if (HelloLabel.visible == 1)
      UW_SetVisible(&HelloLabel,0);
  else
      UW_SetVisible(&HelloLabel,1);
}
```

IV. Initialize:

In this function we will:

1. Add the Icon to the main window
2. Attach OnClick Event of Icon to EnterForm
3. Add the Hello form the main window
4. Attach OnClick Event of Form to ExitForm
5. Add Label to Hello form
6. Add Button to Hello form
7. Attach OnClick Event of Button to Toggle Label

The following code represents the steps needed to achieve the previous mentioned operations

```
void Hello_Init(void)
{

    u8 index;
    u8 x,y;
    /* Add Icon to Main Window           */
    UW_Add(&HelloIcon, UW_ICON, &UWindows, NULL);
    /* Set Position of the Icon  in Main Window */
    UW_SetPosition(&HelloIcon, 10, 60);
    /* Set Iconcaption in Main Window*/
    UW_SetCaption(&HelloIcon, IconCaption, 4);
```

```
    /*Set Click Handler routine to show form */
    UW_SetOnClick(&HelloIcon, HelloEnter);

    /* Add Form to Main Window   and Hide it*/
    UW_Add(&HelloForm, UW_FORM, &UWindows, NULL);
    /* Set Formcaption */
    UW_SetCaption(&HelloForm, IconCaption, 4);
    /* Hide the Form in Main Window  */
    UW_SetVisible(&HelloForm, 0);
    /*Set Click Handler routine to hide the form*/
    UW_SetOnClick(&HelloForm, HelloExit);

    /* Add Label to Hello Form       */
    UW_Add(&HelloLabel,UW_LABEL, &HelloForm, NULL);
    /* Set Formcaption */
    UW_SetCaption(&HelloLabel, LabelCaption, 15);
    UW_SetPosition(&HelloLabel, 5, 80);
    UW_SetSize(&HelloLabel, 115, 20);
    /* Add Btn to Hello Form         */
    UW_Add(&HelloBtn,UW_BUTTON, &HelloForm, NULL);
    /* Set Button  caption */
    UW_SetCaption(&HelloBtn, BtnCaption, 9);
    UW_SetPosition(&HelloBtn, 24, 20);
    UW_SetSize(&HelloBtn, 80, 30);
    /*Set Click Handler routine for the button to
    Toggle Label visibility   */
    UW_SetOnClick(&HelloBtn,HelloBtnClick);
}
```

VI. Integrating Into Application:

For the user to use UWindows two main APIs should be called.

- `void UW_Init(void)`

This is used to initialize the UWindows Library and should be called in Application init routine. It should be called before any other call for UWindows APIs. After calling this function the user can initialization routines for the Uwinwod applications embedded in the Circle_OS

application.

- `enum MENU_code UW_Run(void)`

This function should be called in the Application run routine that is called reguraly by the Circle OS. Most of the periodic handling of the UWindows is done in this function. As the prototype indicates the output of function indicates when to end the application. Therefore it is recommended that this function should be passed as a parameter for the ruturn function i.e.:

```
return (UW_Run());
```

for more details of the implementation please find the attached sample files.