Graphical User Interface - Part 1

In this section, we consider the Graphical User Interface (GUI) facilities offered by Creative Basic.

This includes normal Windows, Dialog Windows, and a range of controls such as Buttons, Text Boxes, Edit Boxes, Rich Edit Boxes, Scroll Bars, Radio Buttons, Check Boxes, List Boxes, Combo Boxes, Toolbars, and Status Windows.

Let's begin **Part 1** with the Windows which hold the controls.

Windows

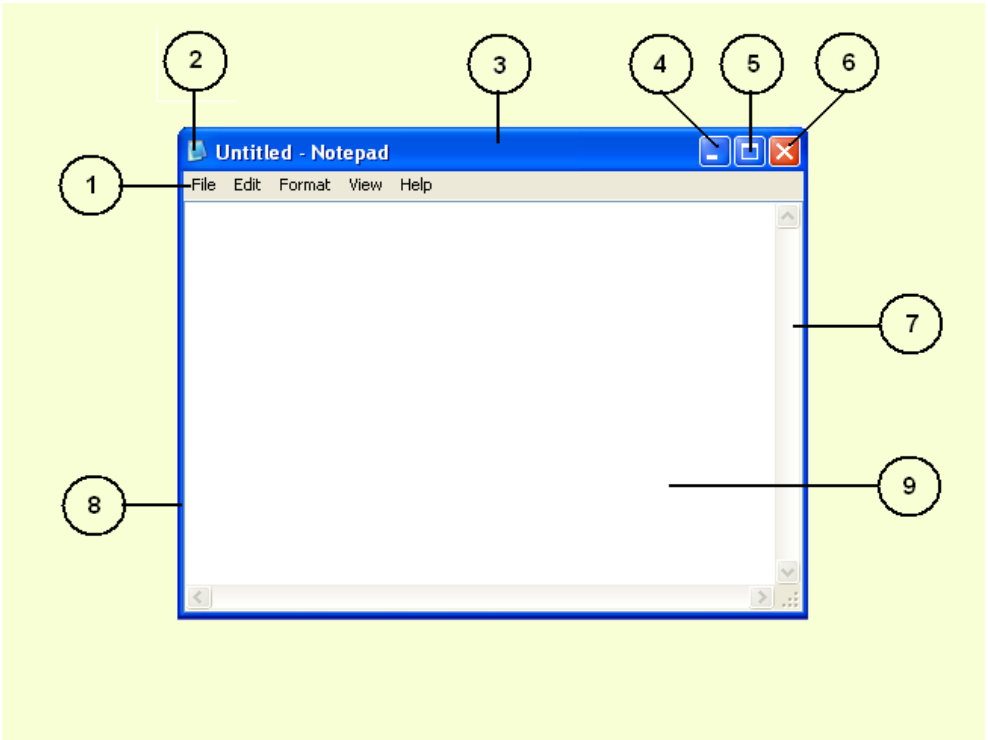
Windows are the user's interface to the computational and graphics power of the computer.

Everyone is familiar with using Windows these days - but their are several types of Window that programmer can use.

Normal Windows

A normal Window is easily recognised. It can take up all of the screen, or just part of it..

Windows vary in style, but usually comprise a number of standard parts, as in the following diagram.



1. Menu bar	2. System Menu	3. Caption Bar
4. Minimise button	5. Maximise button	6. Close button
7. Scroll Bar	8. Border	9. Workspace

Here's the code to create a simple window ..

```
' Normal window
def w:window
def wstyle:int

wstyle = @size|@minbox
' open a window ..
window w,-600,0,600,400,wstyle,0,"Simple Window",messages
setwindowcolor w,rgb(0,60,190)
centerwindow w

control w,"B,Exit,(600-70)/2,310,70,30,0,1"

run = 1

waituntil run = 0
closewindow w
END

sub messages
' routine to handle all the windows messages ..
select @class
   case @idclosewindow
      run = 0
   case @idcontrol
      select @controlID
         case 1
            run = 0
      endselect
endselect
return
```

You will probably recognise this from an earlier chapter of the user guide notes.

Two things to note. In this example, the style only provides for a window that can be resized and minimised. If we needed to provide an option for it to be maximised, the parameter **@maxbox** would have been included.

Also notice that the window 'x' co-ordinate is set to -600 (the width of the window). The reason for this, is to position the window off-screen during the setup period, to avoid screen flashing. If you use this trick, take care to re-position the window afterwards - otherwise it will be off to the left of the screen and invisible - so you can't close it (very embarrassing).

The example window's 'parent' flag is set to '0', since this window has no parent. This parameter is only set when you create a 'Child' window. We'll take a look a this shortly.

Window Style Flags

Here's a list of the style values that can be set for a window ..

Value	Purpose / Style
@SIZE	Creates a window that is re-sizable
@MINBOX	The window has a minimize box
@MAXBOX	The window has a maximize box
@MINIMIZED	Creates a window that is initially minimized
@MAXIMIZED	Creates a window that is initially maximized
@CAPTION	(Default). Creates a window with a caption
@NOCAPTION	Creates a window with no caption bar
@SYSMENU	(Default). Creates a standard system menu
@BORDER	Creates a bordered window. Use with the @NOCAPTION flag
@HSCROLL	Window has a horizontal scroll bar
@VSCROLL	Window has a vertical scroll bar
@MDIFRAME	Creates a window that can contain other windows.
@USEDEFAULT	Child windows can use this for the 'left' parameter to let windows pick a default size for the window
@TOOLWINDOW	Creates a window with a half-sized caption.
@NOAUTODRAW	Allows your program to handle @IDPAINT messages
@BROWSER	Creates a window with an embedded internet explorer window.

Normally, a window will have a caption bar at the top, and this is used to click and drag the window around the screen.

So what happens if you use the **@NOCAPTION** option?
If there's no caption, the window can't be moved - unless we use a little bit of API magic.

The following example program sends a message to the window, making it think a caption does exist. Then when the left mouse button is pressed, the window can be moved around the screen.

In this example, you will also see that the window remains on screen as the topmost window, no matter what other windows exist. The small button on the right closes the window.

No Caption Clock Display

```
' Desktop (dragable) Time Display
' GWS - September 2007 - Creative Basic code

def w:window
def time:string
def flags,false,true,drag,xm,ym:int
def x,y,wid,hgt,xn,yn,scrW,scrH:int

getscreensize scrW,scrH

declare "user32",SetWindowPos(win:window,ins:int,x:int,y:int,wid:int,hgt:int,flags:int),int

window w,(scrW-60)/2,0,60,18,@nocaption,0,"",main
control w,"B,,50,0,10,18,0,1"
setwindowcolor w, 0x555555

setid "SWP_NOMOVE",2
setid "SWP_NOSIZE",1
setid "HWND_TOPMOST",-1
setid "WM_NCLBUTTONDOWN",0xA1
setid "HTCAPTION",2

false = (1 = 2)
true = not false

drag = false

flags = @SWP_NOMOVE|@SWP_NOSIZE
SetWindowPos(w,@HWND_TOPMOST,0,0,0,0,flags)

frontpen w, 0xffff00
setfont w, "Times New Roman",8,400

timeset
starttimer w,10000,1

run = 1

waituntil run = 0
stoptimer w,1
closewindow w
END

SUB main
select @CLASS
   case @IDCLOSEWINDOW
      run = 0
   case @IDCONTROL
      select @CONTROLID
         case 1
            run = 0
      endselect
   case @IDLBUTTONDN
' To Move the Window by dragging it .. (Steve's method)
      SendMessage w, @WM_NCLBUTTONDOWN,@HTCAPTION,0
  case @idtimer
      timeset
endselect
RETURN

sub timeset
' set time in HH.MM format and display it ..
   time = left$(time$,5)
   move w,12,1
   print w,time
return
```

Child Windows

Once you've defined a main window, you can set up one or more 'child' windows within it.

The main window caters for the whole application, and each child window will usually tackle one task within the whole program. A child window can be closed when it's task is done, and re-opened again if required. The main window remains active until it is closed.

If the main window is closed, all it's child windows will close as well.

Here's a simple test program to illustrate a main program with one child window ..

```
' Child window example

def w,child:window
def wstyle:int

wstyle = @size|@minbox
' open main window ..
window w,-600,0,600,400,wstyle,0,"Simple Window",messages
setwindowcolor w,rgb(0,60,190)
centerwindow w
' open child window ..
window child,(600-300)/2,50,300,200,wstyle,w,"Child Window",childmessages
setwindowcolor child,rgb(0,160,190)

control w,"B,Exit,(600-70)/2,310,70,30,0,1"

run = 1

waituntil run = 0
closewindow w
END

sub messages
' routine to handle the main window messages ..
select @class
   case @idclosewindow
      run = 0
   case @idcontrol
      select @controlID
         case 1
            run = 0
      endselect
endselect
return

sub childmessages
' routine to handle the child window messages ..
select @class
   case @idclosewindow
      closewindow child
endselect
return
```

This example program does nothing except display a child window, which is a fully functional window, which can be moved and resized. In this simple example, once you close it, it's gone.

In a working program, it would do something useful, and could be called more than once. It could be used for communicating a small amount of information with the user - like getting a name, offering a choice , or something like that. It's a window within a window.

A Frame Window

This is a window that offers the user a menu of choices of different aspects of the application using a purpose designed window for each subject.

For example, a Morse code program might offer a menu bar with the items, Preferences, Reading, Sending, and Help. There could be sub-menu items such as, Listen to Letters, Listen to Numbers, etc ..

There doesn't have to be a menu bar - you could use controls on the various windows to bring up other child windows.

When the user is finished with that subject, the child window is closed and the user is returned to the main window.

All of the child windows are re-usable, and are automatically closed when the main window closes.

These multiple 'child' windows usually play a more significant role in the application than the simple floating window in our previous example.

It's only a simple bit of code, so the windows do nothing. In a real program, they would offer a significant function such as a table, a graph, a calculation, some data entry, or similar part of the application.

The windows are independent, and can be closed and re-opened as required. Normally, each child window would occupy the whole screen.

There is an alternative approach using the Multiple Document Interface, or MDI facility. (See the user guide and the example 'mdidemo.cba' if you would like to try this).

MDI is not a method I've tried. One disadvantage of this sort of user interface, is that it's easy to 'lose' windows due to overlapping. What you want to see is often hidden.

It's for this reason Microsoft has been phasing out this sort of interface from its own products.

Here's an outline program, which shows a structure using two child windows and an 'about' window.

Child Windows in a Frame Example Program

```
' An example of Child windows in a Frame window

def Frame,c1,c2,c3:window
def i,style,run:int
def wW,wH,bW,bH:int
def textW,textH:int
def a$,b$,c$,title$:string

' set button size ..
bW = 80
bH = 30
' set window size ..
wW = 800
wH = 300

' open main Frame window ..
mainset
' open other child windows ..
otherset

run = 1

waituntil run = 0
closewindow frame                                   :' this also closes all child windows
END

' main window messages ..
sub FrameMessages
   select @class
      case @idclosewindow
         run = 0
      case @idcontrol
         select @controlid
            case 1
' Main Window Exit clicked ..
               run = 0
         endselect
      case @idmenupick
         select @menunum
            case 10
               showwindow c3, @swrestore          :' show the About panel ..
               starttimer c3,2000,1
            case 100                              :' File menu Exit
               run = 0
            case 1
               showwindow c1, @swrestore          :' open the Child 1 Window ..
            case 2
               showwindow c2, @swrestore          :' open the Child 2 Window ..
         endselect
   endselect
return

sub c1messages
' Child Window 1 message handler ..
select @class
   case @idcontrol
      select @controlid
         case 1
            showwindow c1, @swhide
      endselect
endselect
return
```

(Continued)

... continuation

```
sub c2messages
' Child Window 2 message handler ..
select @class
   case @idcontrol
      select @controlid
         case 1
            showwindow c2, @swhide
      endselect
endselect
return

sub c3messages
' process the About box messages ...
select @class
   case @idtimer
' timer used to display the about box ..
      showwindow c3, @swhide
      stoptimer c3,1
endselect
return

sub mainset
' set up the main window ..
style = @minbox
window frame,-wW,0,wW,wH,style,0,"Frame Window Example",FrameMessages
setwindowcolor frame,rgb(0,0,80)

' set intro text ...
frontpen frame, rgb(20,130,255)
drawmode frame,@TRANSPARENT
title$ = "Child Window Example"
setfont frame, "Arial",18,700,0
gettextsize frame, title$, textW, textH
move frame,(wW - textW)/2,30
print frame,title$

' main menu ..
menu frame,"T,File,0,0","I,Exit,0,100"
insertmenu frame,1,"T,Select,0,0","I,Child 1,0,1","I,Child 2,0,2"
insertmenu frame,2,"T,Help,0,0","I,About,0,10"

control frame,"B,Exit,(wW-bW)/2,185,bW,bH,@CTLBTNFLAT,1"
setfont frame, "MS Sans Serif",10,600,0,1
setcontrolcolor frame,1,rgb(100,200,250),rgb(10,10,120)
centerwindow frame

return
```

(Continued)

... continuation

```
sub OtherSet
' set up other child windows ..

' create child window 1 ..
window c1,20,20,200,200,@nocaption,frame,"",c1messages
setwindowcolor c1, rgb(150,150,120)
control c1,"B,Close,(200-bW)/2,150,bW,bH,@CTLBTNFLAT,1"
setfont c1, "MS Sans Serif",10,600,0,1
setcontrolcolor c1,1,rgb(100,200,250),rgb(10,100,120)
setfont c1, "MS Sans Serif",12,600
frontpen c1,rgb(0,0,0)
a$ = "Child 1"
gettextsize c1,a$,textW,textH
move c1,(200-textW)/2,20
print c1, a$
showwindow c1,@swhide                :' hide the Child 1 window until needed

' create child window 2 ..
window c2,575,20,200,200,@nocaption,frame,"",c2messages
setwindowcolor c2, rgb(150,150,120)
control c2,"B,Close,(200-bW)/2,150,bW,bH,@CTLBTNFLAT,1"
setfont c2, "MS Sans Serif",10,600,0,1
setcontrolcolor c2,1,rgb(100,200,250),rgb(10,100,120)
setfont c2, "MS Sans Serif",12,600
frontpen c2,rgb(0,0,0)
a$ = "Child 2"
gettextsize c2,a$,textW,textH
move c2,(200-textW)/2,20
print c2, a$
showwindow c2,@swhide                :' hide the Child 2 window until needed

' create the About window (Child 3)..
window c3,(wW-300)/2,60,300,120,@nocaption,frame,"",c3messages
setwindowcolor c3, rgb(50,150,120)
control c3,"T,,15,15,270,90,@cteditcenter,1"    :' create the About text box ..
setfont c3, "MS Sans Serif",8,500,0,1
setcontrolcolor c3,1,rgb(50,170,255),rgb(0,50,50)
a$ = chr$(10) + "Written in Creative Basic" + chr$(10)
b$ = "Make CBasic your programming tool."
c$ = a$ + "December 2013" + chr$(10) + Chr$(10) + b$
setcontroltext c3,1,c$
showwindow c3,@swhide                :' hide the About box until needed

return
```

Dialog Windows

I recommend that you use normal windows instead of Dialog boxes.

A Dialog box is intended to obtain some necessary data, before an application can continue - like “Enter Password”. It blocks any further user interaction with an application, until the required information is entered.

I’ll demonstrate how this is better achieved using a normal window instead of a Dialog box.

```
' Example using a normal Window instead of a Dialog

def w,d:window
def wstyle,i:int
def textW,textH:int
def a$,pass:string

' open main window ..
mainset
' set up the dialog equivalent window ..
dialset

run = 1

waituntil run = 0
closewindow w
END

sub Mainset
' set up the main application window ..
wstyle = @minbox
window w,-600,0,600,400,wstyle,0,"Dialog Equivalent Window",MainMessages
setwindowcolor w,rgb(0,60,190)
centerwindow w
control w,"B,Exit,(600-100)*2/3,290,100,30,0,1"
control w,"B,Get Password,(600-100)/3,290,100,30,0,2"

return

sub dialset
' set up a small window as a dialog equivalent ..
' not possible with a dialog, has to be done when initialised
window d,0,0,200,120,@NOCAPTION,w,"",dialoghandler
control d,"E,,(200-100)/2,40,100,25,@CTEDITCENTER,1"
control d,"B,OK,(200-50)/2,90,50,20,0,2"
control d,"T,Enter Password,(200-100)/2,10,100,25,0x200|@CTEDITCENTER,3"
setwindowcolor d,rgb(100,160,240)
setcontrolcolor d,1,0,rgb(100,150,180)
setcontrolcolor d,3,0,rgb(100,160,240)
setcontroltext d,1,""
rect d,2,3,196,125,0x0,rgb(100,160,240)    :' a graphic, but not in a Dialog
setsize d,(600-200)/2,80,200,130
showwindow d,@swhide

return
```

(Continued)

... continuation

```
sub MainMessages
select @class
   case @idclosewindow
      run = 0
   case @idcontrol
      select @controlID
         case 1
            run = 0
         case 2                        :' the Get Password button was clicked
            setcontroltext d,1,""
            setfocus d,1
            showwindow d,@swrestore
      endselect
endselect
return

sub dialoghandler
select @CLASS
      case @IDCONTROL
         if (@CONTROLID = 2)      :' the OK button was clicked
            pass = getcontroltext(d,1)
            if (pass = "")        :' check that something has been entered ..
               setfocus d,1       :' otherwise do nothing
               return
            endif
' check for a valid password here - if invalid, display an error message
' assuming the password is valid, hide the dialog window
' and display the password in the main window
            showwindow d,@swhide
            frontpen w, rgb(160,160,255)
            setfont w, "Arial",14,700,0
            a$ = "The Password is: " + pass
            gettextsize w, a$, textW, textH
            move w,(600-textW)/2,80
            print w,a$
            a$ = "You're clear to go .."
            gettextsize w, a$, textW, textH
            move w,(600-textW)/2,130
            print w,a$
' if password is valid, you could hide the get password button at this point.
            showwindow w,@SWHIDE,2              :' hide the get password button
            setsize w,(600-100)/2,290,100,30,1  :' center the Exit button
         endif
endselect

return
```

I'm comfortable with this example - it has none of the snags that can arise when using a Dialog box. Problems with the Tab, ESC and Enter keys for example which usually need workarounds.

Next we'll take a look at the useful Message Box facility.

Message Boxes

Message boxes are very useful things, not only for messages to the user as their name suggest, but for displaying content information of selected variables while testing your program.

Normally, the message box will display a warning message and offer the user a choice of options. The various options return a value enabling the program to carry out the appropriate action.

A message box is easily generated by inserting a message box statement in your program.

MESSAGEBOX window, text, caption {,flags}

The Text, is what you want to say to the user. The caption describes what the message relates to, and the flags are used to show a selection of several buttons and icons which indicate the type of message.

Here is a list of the **Messagebox Button** flags that are available ..

Description	*Hexadecimal Value*	*Integer Value*
OK	**0x00000000**	**0**
OK, and Cancel	**0x00000001**	**1**
Abort, Retry, and Ignore	**0x00000002**	**2**
Yes, No, and Cancel	**0x00000003**	**3**
Yes, and No	**0x00000004**	**4**
Retry, and Cancel	**0x00000005**	**5**
Cancel, Try Again, and Continue	**0x00000006**	**6**

The Button values can be added to Icon values (shown in the following table), to display a desired Icon as well as the buttons.

Here are the Messagebox **Icon Flags** that are available ..

Description	*Hexadecimal Value*	*Integer Value*
Error, or Stop	**0x00000010**	**16**
Question	**0x00000020**	**32**
Exclamation	**0x00000030**	**48**
Information	**0x00000040**	**64**

To set the focus to a desired button left to right, add in one of the following **Default Button**

Description	*Hexadecimal Value*	*Integer Value*
1st button (default)	**0x00000000**	**0**
2nd button	**0x00000100**	**256**
3rd button	**0x00000200**	**512**

You may require the Messagebox to take up the **top** window position, so that it can't be hidden behind some other window ..

Description	*Hexadecimal Value*	*Integer Value*
MessageBox Topmost	**0x00040000**	**262144**

Combining hexadecimal values is easy. You don't need to actually add any values together arithmetically, you just insert each hex digit into the correct position in the one flag value.

For example, to combine **Yes and No Buttons** (**0x00000004**),
With an **Information Icon** (**0x00000040**),
And set the focus to the **NO Button** (second from the left) **(0x00000100)**

The complete flag value then becomes .. **0x00000144**
(Or in integer values (4 + 64 + 256) = 324)

To make the Messagebox the topmost window, just add in the Topmost value (**0x00040000**)
And you get the final hex value .. **0x00040144**

Notice that the hexadecimal values just slip neatly into their own positions. You have to do a bit of arithmetic if you use equivalent the integer values.

Here's a small window using the above example, showing a Messagebox with **Yes, No Buttons**, an **Information Icon**, the focus set to the **Second button** (No), and the Messagebox set to the **Top Window** position ..

```
' MessageBox Example window

def w:window
def wstyle,i:int

wstyle = @minbox
window w,-600,0,600,400,wstyle,0,"MessageBox Example",messages
setwindowcolor w,rgb(0,60,190)
centerwindow w

control w,"B,Exit,(600-70)/2,310,70,30,0,1"

messagebox w,"Delete the file?","Confirm",0x00040144

run = 1

waituntil run = 0
closewindow w
END

sub messages
select @class
   case @idclosewindow
      run = 0
   case @idcontrol
      select @controlID
         case 1
            run = 0
      endselect
endselect
RETURN
```

That completes the discussion of the containing window types.

In Part 2 of the GUI guide, we take a look at the various Controls available in Creative Basic.