Chapter

17

第17章　Visual Fortran 概述

這一章會介紹 Visual Fortran 在 FORTRAN 標準外所擴充的功能，主要分成兩大部分；第一部分會介紹 Visual Fortran 的擴充函式，第二部分會介紹 Visual Fortran 的繪圖功能。

17-1 Visual Fortran 擴充函式

Visual Fortran 中提供了很多讓 FORTRAN 跟作業系統溝通的函式，這些函式都包裝在 MODULE DFPORT 中。呼叫這些函式前，請先確認程式碼中有使用 USE DFPORT 這一行指令。

integer(4) function IARGC()

傳回執行時所傳入的參數個數

subroutine GETARG(n, buffer)

用命令列執行程式時，可以在後面加上一些參數來執行程式，使用 GETARG 可以取出這些參數的內容。

`integer n`	決定要取出哪個參數
`character*(*) buffer`	傳回參數內容

FORTRAN 程式編譯好後，執行程式時可以在命令列後面加上一些額外的參數。假如有一個執行檔為 a.exe，執行時若輸入 a –o –f，在 a 之後的字串都會被當成參數。這時候執行 a –o -f 時，呼叫函數 IARGC 會得到 2，因為總共傳入了兩個參數。呼叫函式 GETARGC(1,buffer)時，字串 buffer=”-o”，也就是第 1 個參數的值。

subroutine GETLOG(buffer)

查詢目前登錄電腦的使用者名稱。

`character*(*) buffer`	傳回使用者名稱

integer(4) function HOSTNAM(buffer)

查詢電腦的名稱，查詢動作成功完成時函數傳回值為 0。buffer 字串長度不夠使用時，傳回值為-1。

`character*(*) buffer`	傳回電腦的名稱

程式執行時，工作目錄是指當開啟檔案時，沒有特別指定目錄位置時會使用的目錄。通常這個目錄就是執行檔的所在位置，在程式進行中可以查詢或改變這個目錄的位置。

integer function GETCWD(buffer)

查詢程式目前的工作目錄位置，查詢成功時函數傳回 0。

integer function CHDIR(dir_name)

把工作目錄轉換到 dir_name 字串所指定的目錄下，轉換成功時傳回 0。

擴充的檔案相關函式，補充了一些原本的缺失。INQURE 指令可以用來查詢檔案資訊，不過它並沒有提供很詳細的訊息，例如檔案大小就沒有辦法使用 INQUIRE 來查詢。

integer(4) function STAT(name, statb)

查詢檔案的資料，結果放在整數陣列 statb 中。查詢成功時函數傳回值為 0。

character*(*) name	所要查詢的檔名。
integer statb(12)	查詢結果，每個元素代表某一個屬性，statb(8)代表檔案長度，單位為 bytes。其它值請參考使用說明。

integer(4) function RENAME(from , to)

改變檔案名稱，改名成功時傳回 0。

character*(*) from	原始檔名
character*(*) to	新檔名

程式執行時，可以經由函式 SYSTEM 再去呼叫另外一個程式來執行。

integer(4) function SYSTEM(command)

讓作業系統執行 command 字串中的命令。

subroutine QSORT(array, len, size, compar)

使用 Quick Sort 演算法把傳入的陣列資料排序。

array	任意型態的陣列
integer(4) len	陣列大小
integer(4) size	陣列中每個元素所佔用的記憶體容量
integer(2), external :: compar	使用者必須自行寫作比較兩筆資料的函式，函式 compar 會自動傳入兩個參數 a、b。當 a 要排在 b 之前時，compar 要傳回負數，a、b 相等時請傳回 0，a 要排在 b 之後時，compar 要傳回正數。

QSORT.F90

```
 1.program main ! 使用 QSORT 函式的示範
 2.  use DFPORT
 3.  implicit none
 4.  integer :: a(5) = (/ 5,3,1,2,4 /)
 5.  integer(2), external :: compareINT
 6.  call QSORT( a, 5, 4, compareINT )
 7.  write(*,*) a
 8.  stop
 9.end program
10.! 要自行提供比較大小用的函數
11.integer(2) function compareINT(a,b)
12.  implicit none
13.  integer a,b
14.  if ( a<b ) then
15.    compareINT=-1 ! a 排在 b 之前
16.  else if ( a==b) then
17.    compareINT=0
18.  else
19.    compareINT=1 ! a 排在 b 之後
20.  end if
```

```
21.  return
22.end function
```

17-2 Visual Fortran 的繪圖功能

第 12 章的 SGL 是筆者提供的繪圖函式庫，Visual Fortran 中另外有提供繪圖功能。Visual Fortran 的繪圖功能不完全是以擴充函式的型態存在，使用它的繪圖功能必須在選擇 Project 型態時，選擇 Standard Graphics 或 QuickWin 模式。

Visual Fortran 提供的繪圖方法，跟 SGL 比較起來算是各有利弊。SGL 使用比較簡單的參數介面，使用效率比較好的 DirectX，支援 Double Buffer 動畫功能，而且並不限制只能在 Visual Fortran 中使用。Visual Fortran 的 QuickWin 及 Standard Graphics 模式在簡單的繪圖使用上會比較方便，不像 SGL 必須自行處理一些視窗訊息。它的繪圖函式功能比較多樣，不過效率會比較差，而且不支援動畫功能。

Standard Graphics 和 QuickWin 模式在繪圖方面的使用方法完全相同，它們都是呼叫相同的函式來繪圖。差別在於 Standard Graphics 只能開啟一個視窗來繪圖、QuickWin 模式則可以開啟多個視窗來繪圖。QuickWin 模式下可以有選單及對話窗的功能，Standard Graphics 則不行。Standard Graphics 模式的程式碼可以原封不動直接轉換到 QuickWin 模式下使用，但是 QuickWin 的程式碼並不一定可以直接拿到 Standard Graphics 模式下使用。

17-2-1 基本繪圖功能示範

這個小節沒有什麼新的概念，只會示範 Visual Fortran 繪圖函式的使用方法。直接來看一個範例程式。這個程式會在螢幕上畫出一條斜線、一個方形及一個橢圓。

編譯程式時，請選擇 Fortran Standard Graphics or QuickWin Application 這個模式。這個程式可以在 Standard Graphics 或 QuickWin 模式下使用。開啟好 Project 後，再把下面的程式加入 Project 中來編譯。

PLOT.F90

```
 1.! 簡單的繪圖示範
 2.! By Perng 1997/9/19
 3.program Plot_Demo
 4.! 使用 Visual Fortran 的繪圖功能時需要 module dflib
 5.use DFLIB
 6.implicit none
 7.  type(xycoord) :: t
 8.  integer :: result
 9.  call MoveTo(10,10,t)  ! 把目前繪圖的位置移動到座標(10,10)
10.  result=LineTo(100,50) ! 從(10,10)到(100,50)間繪一條直線
11.  ! 畫一個左上角為(110,10), 右下角為(150,50)的實心方形
12.  result=Rectangle( $GFILLINTERIOR, 110,10, 150, 50 )
```

```
13.  ! 畫一個可以放入在(10,60)--(150,100)方形當中的空心橢圓
14.  result=Ellipse($GBORDER, 10, 60, 150, 100)
15.end program Plot_Demo
```

使用 Standard Graphics 模式時，會出現一個繪圖視窗來畫圖。使用 QuickWin 模式時，除了繪圖視窗外，還有內定的選單可以使用。File 選單中的 Print 可以把圖形印出，Save 可以把繪圖結果儲存成*.BMP 圖檔。

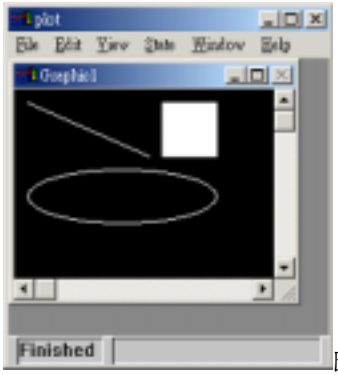

圖 **錯誤! 所指定的樣式的文字不存在文件中。**-1

PLOT.F90 用 QuickWin 模式編譯的結果

使用 Visual Fortran 的繪圖功能時，開啟視窗的工作是自動完成的。程式碼只需要直接呼叫繪圖函式就可以進行繪圖，下面對程式中所使用的繪圖函式做一些介紹：

subroutine MoveTo(x,y,t)

使用這個副程式時，要先把螢幕想像成一張畫紙，程式會使用一隻畫筆在螢幕上畫畫。MoveTo(x,y,t)可以把這隻畫筆移動到畫紙上的(x,y)座標處，參數 t 則會傳回移動之前的畫筆所在位置（這個參數其實沒有什麼用，不過既然有規定就一定要把它放入）。

請注意，原點(0,0)是位在視窗的左上角，x 座標軸向右為正，y 座標軸向下為正。

integer(2) function LineTo(x,y)

這個函數可以把畫筆從目前的位置到(x,y)處畫一條直線。傳回值如果不為 0，代表函數運作不正常。

integer(2) function Rectangle(control, x1, y1, x2, y2)

這個函數可以在(x1,y1)、(x2,y2)兩個端點間畫出一個方形。control 值可以用來設定是要畫出一個實心方形或是只有外框而已。在範例中把 control 的值用 $GFILLINTERIOR 來代入，表示要畫實心。$GFILLINTERIOR 定義在 MODULE DFLIB 裏面。

integer(2) function Ellipse(control,x1,y1,x2,y2)

這個函數會在(x1,y1)、(x2,y2)兩端點間所形成的矩形中畫橢圓。control 的意義同上，在範例中使用$GBORDER，代表只畫出外框。

再來看一個範例，它會畫出 SIN 函數的圖形。

SIN.F90

```
 1.! sin 函數的繪圖示範
 2.program Plot_Sine
 3. use DFLIB
 4.implicit none
 5.  integer, parameter :: lines=500   ! 用多少線段來畫函數曲線
 6.  real(kind=8), parameter :: X_Start=-5.0   ! x 軸最小範圍
 7.  real(kind=8), parameter :: X_End=5.0      ! x 軸最大範圍
 8.  real(kind=8), parameter :: Y_Top=2.0      ! y 軸最大範圍
 9.  real(kind=8), parameter :: Y_Bottom=-2.0  ! y 軸最小範圍
10.  integer :: result        ! 取回繪圖函數運作狀態
11.  integer(kind=2) :: color  ! 設定顏色用
12.  real(kind=8) :: step     ! 迴圈的增量
13.  real(kind=8) :: x,y      ! 繪圖時使用,每條小線段都連接
14.  real(kind=8) :: NewX,NewY ! (x,y)及(NewX,NewY)
15.  real(kind=8), external :: f ! 待繪圖的函數
16.  type(wxycoord) :: wt     ! 傳回上一次的虛擬座標位置
17.  type(xycoord)  :: t      ! 傳回上一次的實際座標位置
18.
19.  ! 設定虛擬座標範圍大小
20.  result=SetWindow( .true. , X_Start, Y_Top, X_End, Y_Bottom )
21.  ! 用索引值的方法來設定顏色
22.  result=SetColor(2)   ! 內定的 2 號是應該是綠色
23.  call MoveTo(10,20,t) ! 移動畫筆到視窗的(10,20)
24.  call OutGText("f(x)=sin(x)")  ! 寫出內容
25.  ! 使用全彩 RGB 0-255 的 256 種色階來設定顏色
26.  color=RGBToInteger(255,0,0)       ! 把控制 RGB 的三個值濃縮到 color 中
27.  result=SetColorRGB(color)        ! 利用 color 來設定顏色
28.
29.  call MoveTo_W(X_Start,0.0_8,wt)   ! 畫 X 軸
30.  result=LineTo_W(X_End,0.0_8)     !
31.  call MoveTo_W(0.0_8,Y_Top,wt)    ! 畫 Y 軸
32.  result=LineTo_W(0.0_8,Y_Bottom)   !
33.
34.  step=(X_End-X_Start)/lines       ! 計算小線段間的 X 間距
35.  ! 參數#FF0000 是使用 16 進位的方法來表示一個整數
36.  result=SetColorRGB(#FF0000)
37.  ! 開始繪製小線段們
38.  do x=X_Start,X_End-step,step
39.    y=f(x)         ! 線段的左端點
40.    NewX=x+step
41.    NewY=f(NewX)    ! 線段的右端點
42.    call MoveTo_W(x,y,wt)
43.    result=LineTo_W(NewX,NewY)
44.  end do
45.  ! 設定程式結束後,視窗會繼續保留
46.  result=SetExitQQ(QWIN$EXITPERSIST)
47.end
48.! 所要繪圖的函數
49.real(kind=8) function f(x)
50.implicit none
51.  real(kind=8) :: x
52.  f=sin(x)
53.  return
54.end function f
```

這個程式會以目前 Windows 解析度的大小來開啟繪圖視窗。程式執行後只能夠看到視窗的一小部分，讀者可以試著按下 Alt+Enter 來把視窗放大成全螢幕大小，不然看不到全部的圖形。使用 QuickWin 模式時，並不像使用 SGL 時一樣可以得到視窗大小改變的訊息，所以 QuickWin 下的虛擬座標是對固定解析度來對應，不會隨著視窗大小改變而縮放圖形。

把這個程式中新使用的繪圖函式做一個介紹

integer(2) function SetWindow(invert, x1,y1, x2,y2)

用來設定虛擬座標，invert 的值是用來指定 Y 軸向上為正或為負。invert=.true.時向上為正，invert=.false.時向下為負。(x1,y1)、(x2,y2)則使用雙精確度浮點數來定義繪圖範圍兩端的虛擬座標值。

integer(2) function SetColor(index)

使用索引值的方法來設定所要使用的顏色。

subroutine OutGText(text)

呼叫這個副程式可以在目前畫筆的位置上寫出 text 字串。

integer(4) function RGBToInteger(R,G,B)

前面有提過，全彩模式中，RGB 三種色光可以各自有 256 種色階變化，所以一個顏色需要使用 3 bytes 的空間來做記錄。但是 FORTRAN 中通常一個整數使用 4 bytes 的空間來做記錄，如果分別使用 3 個變數來記錄一個顏色值會很浪費空間，因為這 3 個變數都只會使用到最低的 8 個位元，其它地方都不會使用。

這個函數可以重新組合 R G B 這三個傳入的整數，取出這三個整數中最低 8 個位元的數值，把它重新安排到傳回值 color 中。color 中的第 0~7 bits 用存放原本的 R，8~15 bits 會存放原本的 G，16~23 bits 會存放原本的 B，剩下第 24~31 bits 的空間則不會使用。

integer(2) function SetColorRGB(color)

用 RGB 方法來設定顏色，參數 color 中的第 0~7 個 bits 用來設定紅光，第 8~15 bits 用來設定綠光，第 16~23 bits 用來設定藍光，其它位元不使用。在範例當中曾使用 SetColorRGB(#FF0000)來設定顏色，在 Visual Fortran 中以#符號開頭的數字，代表一個 16 進位的數字。

16 進位數值在 0~9 時和 10 進位數字相同，但是接在 9 下面的數字為 A、B、C、D、E、F。其中 A 等於 10 進位的 10，B=11，C=12.....，同理可推得 10_{16}=16，FF_{16}=255。使用 16 進位的系統可以比較容易來操作這種需要控制到位元內容的數值。正規的 FORTRAN 90 寫法應該用 Z"FF0000"來設定 16 進位的數值，第 5 章的最後一節有介紹這個方法。在此順便示範一下 Visual Fortran 的擴充語法。

17-3 互動功能

這一節要介紹 QuickWin 模式下，使用者和程式間的互動功能。滑鼠及選單、對話窗等等的工具，都是本節所要討論的範圍。

17-3-1 傳統的鍵盤輸入

如果讀者想把原先發展好的程式，轉換到 QuickWin 模式下來使用，還是可以使用 READ/WRITE 指令來輸出入。

下面的範例程式會根據使用者的選擇來畫出函數圖形。程式執行後，會出現兩個小視窗，一個視窗會負責讀取鍵盤的輸入，另一個視窗則負責繪圖。

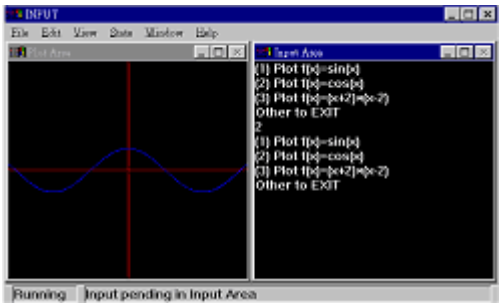

圖 **錯誤! 所指定的樣式的文字不存在文件中。**-2

這個程式開啟了兩個視窗，分別用來做繪圖及輸入的工作。讀者可以注意到繪圖部分的程式碼和上一節的程式一模一樣，不過這個程式把繪圖視窗縮小了，所以不經過放大就可以看到全部的圖形。因為繪圖座標系是使用虛擬座標，所以圖形的相對位置不會因視窗解析度而改變。

INPUT.F90 (QuickWin 模式)

```
 1.! Interactive 的繪圖示範
 2.! By Perng 1997/9/20
 3.program Plot_Sine
 4.use DFLIB
 5.implicit none
 6.  logical :: result
 7.  integer :: input
 8.  type(WindowConfig) :: wc
 9.  real(kind=8), external :: f1,f2,f3    ! 所要畫的函數
10.  ! 開啟輸入用的視窗
11.  open(unit=5,file='user',iofocus=.true.)
12.  ! -1 代表由程式自行去做決定
13.  wc.numxpixels=300 ! 視窗的寬
14.  wc.numypixels=300 ! 視窗的高
15.  wc.numtextcols=-1 ! 每行可容納的文字
16.  wc.numtextrows=-1 ! 可以有幾列文字
17.  wc.numcolors=-1   ! 可以使用的顏色
18.  wc.title="Input Area"C ! 視窗的標題文字
19.  wc.fontsize=-1    ! 所使用的文字大小
```

```
20.  ! 根據 wc 中所定義的資料來重新設定視窗大小
21.  result=SetWindowConfig(wc)
22.  result=DisplayCursor($GCURSORON) ! 顯現出游標
23.  ! 開啟繪圖所要使用的視窗
24.  open(unit=10,file='user',iofocus=.true.)
25.  wc.numxpixels=300 ! 視窗的寬
26.  wc.numypixels=300 ! 視窗的高
27.  ! -1 代表讓程式自行去做決定
28.  wc.numtextcols=-1! 每行容量的文字
29.  wc.numtextrows=-1! 可以有幾列文字
30.  wc.numcolors=-1  ! 使用多少顏色
31.  wc.title="Plot Area"C ! 視窗的標題
32.  wc.fontsize=-1
33.  ! 根據 wc 中所定義的資料來重新設定視窗大小
34.  result=SetWindowConfig(wc)
35.  ! 程式自動按下選單中 Windows 的 Tile 指令, 使兩個視窗之間
36.  ! 不會互相重疊
37.  result=ClickMenuQQ(QWIN$TILE)
38.  input=1 ! 隨便給一個合理的值, 不然不能進入下面的迴圈
39.  ! 把輸入使用的視窗設為可以被輸入的狀態, 5 就是第一次開啟
40.  ! 繪圖視窗時所用的 unit 值
41.  result=FocusQQ(5)
42.  do while( input>0 .and. input<4 )
43.    write(5,*) '(1) Plot f(x)=sin(x)'
44. write(5,*) '(2) Plot f(x)=cos(x)'
45. write(5,*) '(3) Plot f(x)=(x+2)*(x-2)'
46. write(5,*) 'Other to EXIT'
47. read(5,*) input
48. result=SetActiveQQ(10) ! 把繪圖指令指定到繪圖視窗的代碼上
49. ! 根據 input 來決定要畫出那一個函數
50. select case(input)
51. case (1)
52.   call Draw_Func(f1)
53. case (2)
54.   call Draw_Func(f2)
55. case (3)
56.   call Draw_Func(f3)
57. end select
58.  end do
59.  ! 設定主程式碼結束後,視窗會自動關閉
60.  result=SetExitQQ(QWIN$EXITNOPERSIST)
61.end program Plot_Sine
62.
63.subroutine Draw_Func(func)
64.use DFLIB
65.implicit none
66.  integer, parameter :: lines=500    ! 用多少線段來畫函數曲線
67.  real(kind=8), parameter :: X_Start=-5.0 ! x 軸最小範圍
68.  real(kind=8), parameter :: X_End=5.0        ! x 軸最大範圍
69.  real(kind=8), parameter :: Y_Top=5.0        ! y 軸最大範圍
70.  real(kind=8), parameter :: Y_Bottom=-5.0    ! y 軸最小範圍
71.  integer :: result         ! 取回繪圖函數運作狀態
72.  integer(kind=2) :: color  ! 設定顏色用
73.  real(kind=8) :: step      ! 迴圈的增量
74.  real(kind=8) :: x,y       ! 繪圖時使用,每條小線段都連接
75.  real(kind=8) :: NewX,NewY ! (x,y)及(NewX,NewY)
76.  real(kind=8), external :: func ! 待繪圖的函數
77.  type(wxycoord) :: wt      ! 傳回上一次的虛擬座標位置
78.  type(xycoord)  :: t       ! 傳回上一次的實際座標位置
79.
80.  call ClearScreen($GCLEARSCREEN) ! 清除螢幕
81.  ! 設定虛擬座標範圍大小
82.  result=SetWindow( .true. , X_Start, Y_Top, X_End, Y_Bottom )
83.  ! 用索引值的方法來設定顏色
```

```
 84.  result=SetColor(2)   ! 內定的 2 號是應該是綠色
 85.  call MoveTo(10,20,t) ! 移動畫筆到視窗的(10,20)
 86.
 87.  ! 使用全彩 RGB 0-255 的 256 種色階來設定顏色
 88.  color=RGBToInteger(255,0,0)    ! 把控制 RGB 的三個值濃縮到 color 中
 89.  result=SetColorRGB(color) ! 利用 color 來設定顏色
 90.
 91.  call MoveTo_W(X_Start,0.0_8,wt)     ! 畫 X 軸
 92.  result=LineTo_W(X_End,0.0_8)        !
 93.  call MoveTo_W(0.0_8,Y_Top,wt)       ! 畫 Y 軸
 94.  result=LineTo_W(0.0_8,Y_Bottom)     !
 95.  step=(X_End-X_Start)/lines     ! 計算小線段間的 X 間距
 96.  ! 參數#FF0000 是使用 16 進位的方法來表示一個整數
 97.  result=SetColorRGB(#FF0000)
 98.  ! 開始繪製小線段
 99.  do x=X_Start,X_End-step,step
100.    y=func(x)       ! 線段的左端點
101. NewX=x+step
102. NewY=func(NewX)   ! 線段的右端點
103. call MoveTo_W(x,y,wt)
104. result=LineTo_W(NewX,NewY)
105.  end do
106.  ! 設定程式結束後,視窗會繼續保留
107.  result=SetExitQQ(QWIN$EXITPERSIST)
108.end subroutine Draw_Func
109.! 所要繪圖的函數
110.real(kind=8) function f1(x)
111.implicit none
112.  real(kind=8) :: x
113.  f1=sin(x)
114.  return
115.end function f1
116.real(kind=8) function f2(x)
117.implicit none
118.  real(kind=8) :: x
119.  f2=cos(x)
120.  return
121.end function f2
122.real(kind=8) function f3(x)
123.implicit none
124.  real(kind=8) :: x
125.  f3=(x+2)*(x-2)
126.  return
127.end function f3
```

程式碼中已經有相當份量的註解，在此只對一些重點做說明。QuickWin 模式下的程式，開啟一個名稱為 USER 的檔案時，就會開啟一個新的子視窗；因為這個視窗永遠只存在於程式視窗範圍中，不會跑到外面，所以叫子視窗。

開啟 USER 檔案時所使用的 UNIT 值，會成為這個視窗的代號。在 QuickWin 中，使用 WRITE/READ 時，只要把輸出入位置指定到子視窗中，就可以對某個子視窗來做讀寫。用這個方法來做輸出入，可以讓舊程式做最少的更改就具備繪圖功能。

還有一點要注意的，這個程式開啟了兩個子視窗，所以在繪圖時，要指定繪圖的動作要畫在哪個視窗上，這個工作是由函式 SetActiveQQ 來完成。

17-3-2 滑鼠的使用

直接由範例程式來學習滑鼠的使用方法。這個範例程式模擬了一隻畫筆，按下滑鼠左鍵畫筆就會落下，在螢幕上畫出一個紅色小點。螢幕左上角還會顯示出目前滑鼠在視窗中的位置。

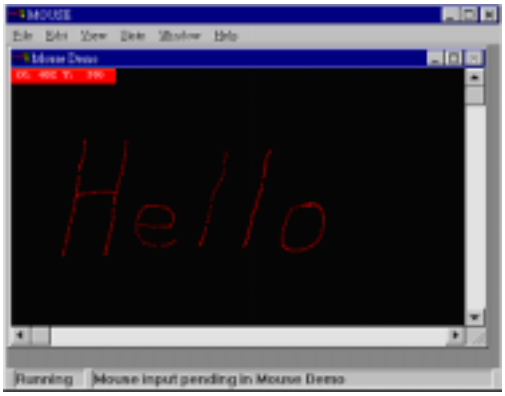

圖 **錯誤! 所指定的樣式的文字不存在文件中。**-3

MOUSE.F90 (QuickWin 模式)

```
1.! 處理滑鼠事件的函式
2.module MouseEvent
3.use DFLIB
4.implicit none
5.Contains
6.  ! 滑鼠在視窗中每移動一次,就會呼叫這個函式
7.  subroutine ShowLocation(iunit, ievent, ikeystate, ixpos, iypos)
8.  implicit none
9.    integer :: iunit       ! 滑鼠所在的視窗的 unit 值
10.   integer :: ievent      ! 滑鼠發生的訊息碼
11.   integer :: ikeystate   ! 進入這個函式時,其它控制鍵的狀態
12.   integer :: ixpos,iypos  ! 滑鼠在視窗中的位置
13.   type(xycoord) :: t
14.   integer :: result
15.   character(len=15) :: output ! 設定輸出的字串
16.
17.   result=SetActiveQQ(iunit)      ! 把繪圖工作指向這個視窗
18.   write(output,100) ixpos,iypos   ! 把滑鼠所在位置的訊息寫入 output
19.100 format("(X:",I4," Y:",I4,")")   !
20.   result=SetColorRGB(#1010FF)
21.   result=Rectangle($GFILLINTERIOR,0,0,120,18) ! 畫一個實心方形
22.   result=SetColorRGB(#FFFFFF)
23.   call MoveTo( 4,2,t)
24.   call OutGText(output)   ! 寫出訊息
25.   ! 如果滑鼠移動時, 左鍵同時被按下, 會順便畫出一個點.
26.   if ( ikeystate==MOUSE$KS_LBUTTON ) then
27.     result=SetColorRGB(#0000FF)
28.     result=SetPixel(ixpos,iypos)
29.   end if
30.   return
31. end subroutine
32. ! 滑鼠右鍵按下時, 會執行這個函式
33. subroutine MouseClearScreen(iunit, ievent, ikeystate, ixpos, iypos )
34. implicit none
35.   integer :: iunit       ! 滑鼠所在的視窗的 unit 值
36.   integer :: ievent      ! 滑鼠發生的訊息碼
```

```
37.    integer :: ikeystate    ! 進入這個函式時,其它控制鍵的狀態
38.    integer :: ixpos,iypos  ! 滑鼠在視窗中的位置
39.    type(xycoord) :: t
40.    integer :: result
41.
42.    result=SetActiveQQ(iunit)       ! 把繪圖動作設定在滑鼠所在視窗上
43.    call ClearScreen($GCLEARSCREEN) ! 清除整個螢幕
44.
45.    return
46.  end subroutine
47.end module
48.
49.program Mouse_Demo
50.use DFLIB
51.use MouseEvent
52.implicit none
53.  integer :: result
54.  integer :: event
55.  integer :: state,x,y
56.
57.  result=AboutBoxQQ("Mouse Draw Demo\r By Perng 1997/09"C)
58.  ! 開啟視窗
59.  open( unit=10, file='user', title='Mouse Demo', iofocus=.true. )
60.  ! 使用字形前, 一定要呼叫 InitializeFonts
61.  result=InitializeFonts()
62.  ! 選用 Courier New 的字形在視窗中來輸出
63.  result=setfont('t''Courier New''h14w8')
64.  call ClearScreen($GCLEARSCREEN)   ! 先清除一下螢幕
65.  ! 設定滑鼠移動或按下左鍵時, 會呼叫 ShowLocation
66.  event=ior(MOUSE$MOVE,MOUSE$LBUTTONDOWN)
67.  result=RegisterMouseEvent(10, event, ShowLocation)
68.  ! 設定滑鼠右鍵按下時, 會呼叫 MouseClearScreen
69.  event=MOUSE$RBUTTONDOWN
70.  result=RegisterMouseEvent(10, event, MouseClearScreen )
71.  ! 把程式放入等待滑鼠輸入的狀態
72.  do while(.true.)
73.    result=WaitOnMouseEvent( MOUSE$MOVE .or. MOUSE$LBUTTONDOWN .or.&
74.    MOUSE$RBUTTONDOWN, state, x, y )
75.  end do
76.end program
```

這個程式使用的觀念，有點類似寫作 SGL 程式的方法。讀者也許會覺得很奇怪，為什麼主程式最後要進入一個無窮迴圈中？為什麼寫了一些處理滑鼠訊息的函式，卻沒有看到程式碼去呼叫它，但是它卻還是會被執行？

程式碼第 72～75 行的部分，用迴圈來等待 Windows 作業系統所傳遞的滑鼠訊息，這就是在第 73 行呼叫 WaitOnMouseEvent 的目的。Windows 作業系統在暗地裏會偷偷塞給應用程式許多訊息，只是有很多訊息會被應用程式忽略。這個程式會處理跟滑鼠相關的訊息。

在 QuickWin 的程式中，如果按下選單 Help 的 About 選項，會出現一個 About 視窗。按下選單 File 中的 Save 選項時，可以把畫面儲存成一個圖檔。這幾個處理訊息的程式碼，都由 Visual Fortran 事先準備好。程式可以多增加幾個處理訊息的程式碼，來增加互動能力。

MOUSE.F90 中，增加了處理滑鼠移動及按下滑鼠左、右鍵訊息的程式。程式碼第 66、67 兩行會設定當滑鼠在代號為 10 的視窗中移動、或是按下左鍵時，會自動呼叫副程式 ShowLocation 來執行。

```
66.  event=ior(MOUSE$MOVE,MOUSE$LBUTTONDOWN)
67.  result=RegisterMouseEvent(10, event, ShowLocation)
```

第 69、70 這兩行程式碼，則會設定當滑鼠右鍵按下時，會自動呼叫副程式 MouseClearScreen。

```
69.  event=MOUSE$RBUTTONDOWN
70.  result=RegisterMouseEvent(10, event, MouseClearScreen )
```

讀者所看到的一些 MOUSE$….開頭的奇怪數值，都是宣告在 MODULE DFLIB 中的常數。這些數值都是滑鼠訊息的代碼，滑鼠可以產生出下列的訊息：

MOUSE$LBUTTONDOWN	按下左鍵
MOUSE$LBUTTONUP	放開左鍵
MOUSE$LBUTTONDBLCLK	左鍵連按兩下
MOUSE$RBUTTONDOWN	按下右鍵
MOUSE$RBUTTONUP	放開右鍵
MOUSE$RBUTTONDBLCLK	右鍵連按兩下
MOUSE$MOVE	滑鼠移動

RegisterMouseEvent 需要 3 個參數，第 1 個參數設定的是:「需要攔截訊息的視窗代碼」，第 2 個參數設定所要處理的滑鼠訊息，第 3 個參數則傳入一個副程式的名字，指定在收到訊息時，會自動執行這個副程式。處理滑鼠訊息的副程式被呼叫時，會傳入下列的參數：

subroutine MouseEventHandleRoutine(iunit,ievent, ikeystate, ixpos, iypos)

integer iunit	得到滑鼠訊息的視窗代碼
integer ievent	滑鼠訊息代碼
integer ikeystate	其它功能鍵的狀態
integer ixpos	滑鼠在視窗中 X 軸的位置
integer iypos	滑鼠在視窗中 Y 軸的位置

要寫作滑鼠訊息處理函式時，一定要遵照這些參數型態來宣告參數，不然在程式執行時會發生錯誤。注意：這種型態錯誤在編譯程式時不會被發現，一定要程式設計師自己小心才行。

根據筆者實驗的結果，範例程式在主程式最後的迴圈，拿掉後一樣可以正常執行程式，程式還是可以得到滑鼠訊息，但是建議不要這麼做。因為原本的做法比較「正規」，而且如果在程式中新增選單功能，這個迴圈就不能省略。

17-3-3 選單的使用

QuickWin 模式下的程式，不需要程式設計師特別設計，先天上就都具備選單功能。這個小節會介紹改變選單預設內容的方法。

在程式中使用選單的方法，和使用滑鼠的原理大致相同。只要先設計好選單內容，再設定按下選項時，會自動執行那個函式就好了。下面的範例程式可以讓使用者從選單中選擇要畫出 SIN(x)或 COS(x)的圖形。

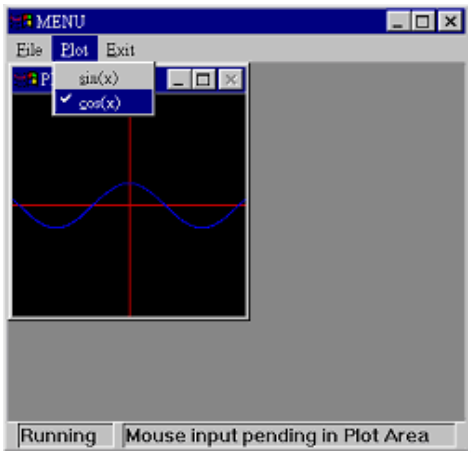

圖 **錯誤! 所指定的樣式的文字不存在文件中。**-4

這個程式和上一個範例程式很類似，只不過把輸入改由選單來做。

MENU.F90

```
 1.! 使用選單示範
 2.! By Perng 1997/9/22
 3.program Menu_Demo
 4.use DFLIB
 5.implicit none
 6.  type(windowconfig) :: wc
 7.  integer :: result
 8.  integer :: i,ix,iy
 9.  wc.numxpixels=200 ! 視窗的寬
10.  wc.numypixels=200 ! 視窗的高
11.  ! -1 代表讓程式自行去做決定
12.  wc.numtextcols=-1 ! 每行容量的文字
13.  wc.numtextrows=-1 ! 可以有幾列文字
14.  wc.numcolors=-1   ! 使用多少顏色
15.  wc.title="Plot Area"C ! 視窗的標題
16.  wc.fontsize=-1
17.  ! 根據 wc 中所定義的資料來重新設定視窗大小
18.  result=SetWindowConfig(wc)
19.  ! 把程式放入等待滑鼠訊息的狀態
20.  do while (.TRUE.)
21.    i = waitonmouseevent(MOUSE$RBUTTONDOWN, i, ix, iy)
22.  end do
23.end program
24.!
25.! 程式會自動執行這個函數, 它會設定視窗的外觀
26.!
27.logical(kind=4) function InitialSettings()
28.use DFLIB
29.implicit none
30.  logical(kind=4) :: result
```

```
31.  type(qwinfo) :: qw
32.  external PlotSin,PlotCos
33.
34.  ! 設定整個視窗程式一開始出現的位置及大小
35.  qw.type=QWIN$SET
36.  qw.x=0
37.  qw.y=0
38.  qw.h=400
39.  qw.w=400
40.  result=SetWSizeQQ(QWIN$FRAMEWINDOW,qw)
41.  ! 組織第一組選單
42.  result=AppendMenuQQ(1,$MENUENABLED,'&File'C,NUL)
43.  result=AppendMenuQQ(1,$MENUENABLED,'&Save'C,WINSAVE)
44.  result=AppendMenuQQ(1,$MENUENABLED,'&Print'C,WINPRINT)
45.  result=AppendMenuQQ(1,$MENUENABLED,'&Exit'C,WINEXIT)
46.  ! 組織第二組選單
47.  result=AppendMenuQQ(2,$MENUENABLED,'&Plot'C,NUL)
48.  result=AppendMenuQQ(2,$MENUENABLED,'&sin(x)'C,PlotSin)
49.  result=AppendMenuQQ(2,$MENUENABLED,'&cos(x)'C,PlotCos)
50.  ! 組織第三組選單
51.  result=AppendMenuQQ(3,$MENUENABLED,'&Exit'C,WINEXIT)
52.
53.  InitialSettings=.true.
54.  return
55.end function InitialSettings
56.!
57.! 畫 sin 的副程式
58.!
59.subroutine PlotSin(check)
60.use DFLIB
61.implicit none
62.  logical(kind=4) :: check
63.  real(kind=8), external :: f1
64.  integer :: result
65.  ! 在第二組選單的第一個選項,也就是 sin 的前面打個勾
66.  result=ModifyMenuFlagsQQ(2,1,$MENUCHECKED)
67.  ! 把選項 cos 前的勾取消
68.  result=ModifyMenuFlagsQQ(2,2,$MENUUNCHECKED)
69.  call Draw_Func(f1)
70.  return
71.end subroutine
72.!
73.! 畫 cos 的副程式
74.!
75.subroutine PlotCos(check)
76.use DFLIB
77.implicit none
78.  logical(kind=4) :: check
79.  real(kind=8), external :: f2
80.  integer :: result
81.  check=.true.
82.  ! 把選項 sin 前的勾取消
83.  result=ModifyMenuFlagsQQ(2,1,$MENUUNCHECKED)
84.  ! 在選項 cos 前打個勾
85.  result=ModifyMenuFlagsQQ(2,2,$MENUCHECKED)
86.  call Draw_Func(f2)
87.  return
88.end subroutine
89.!
90.! 畫出所傳入的函數圖形來
91.!
92.subroutine Draw_Func(func)
93.use DFLIB
94.implicit none
95.  integer, parameter :: lines=500   ! 用多少線段來畫函數曲線
96.  real(kind=8), parameter :: X_Start=-5.0   ! x 軸最小範圍
97.  real(kind=8), parameter :: X_End=5.0      ! x 軸最大範圍
98.  real(kind=8), parameter :: Y_Top=5.0      ! y 軸最大範圍
```

```
 99.  real(kind=8), parameter :: Y_Bottom=-5.0  ! y 軸最小範圍
100.  integer :: result        ! 取回繪圖函數運作狀態
101.  integer(kind=2) :: color  ! 設定顏色用
102.  real(kind=8) :: step      ! 迴圈的增量
103.  real(kind=8) :: x,y       ! 繪圖時使用,每條小線段都連接
104.  real(kind=8) :: NewX,NewY ! (x,y)及(NewX,NewY)
105.  real(kind=8), external :: func ! 待繪圖的函數
106.  type(wxycoord) :: wt      ! 傳回上一次的虛擬座標位置
107.  type(xycoord)  :: t       ! 傳回上一次的實際座標位置
108.
109.  call ClearScreen($GCLEARSCREEN) ! 清除螢幕
110.  ! 設定虛擬座標範圍大小
111.  result=SetWindow( .true. , X_Start, Y_Top, X_End, Y_Bottom )
112.  ! 用索引值的方法來設定顏色
113.  result=SetColor(2)   ! 內定的 2 號應該是綠色
114.  call MoveTo(10,20,t) ! 移動畫筆到視窗的(10,20)
115.
116.  ! 使用全彩 RGB 0-255 的 256 種色階來設定顏色
117.  color=RGBToInteger(255,0,0)       ! 把控制 RGB 的三個值濃縮到 color 中
118.  result=SetColorRGB(color)         ! 利用 color 來設定顏色
119.
120.  call MoveTo_W(X_Start,0.0_8,wt)   ! 畫 X 軸
121.  result=LineTo_W(X_End,0.0_8)      !
122.  call MoveTo_W(0.0_8,Y_Top,wt)     ! 畫 Y 軸
123.  result=LineTo_W(0.0_8,Y_Bottom)   !
124.
125.  step=(X_End-X_Start)/lines        ! 計算小線段間的 X 間距
126.  ! 參數#FF0000 是使用 16 進位的方法來表示一個整數
127.  result=SetColorRGB(#FF0000)
128.
129.  ! 開始繪製小線段
130.  do x=X_Start,X_End-step,step
131.    y=func(x)       ! 線段的左端點
132.    NewX=x+step
133.    NewY=func(NewX) ! 線段的右端點
134.    call MoveTo_W(x,y,wt)
135.    result=LineTo_W(NewX,NewY)
136.  end do
137.
138.  ! 設定程式結束後,視窗會繼續保留
139.  result=SetExitQQ(QWIN$EXITPERSIST)
140.end subroutine Draw_Func
141.!
142.! 所要繪圖的函數
143.!
144.real(kind=8) function f1(x)
145.implicit none
146.  real(kind=8) :: x
147.  f1=sin(x)
148.  return
149.end function f1
150.
151.real(kind=8) function f2(x)
152.implicit none
153.  real(kind=8) :: x
154.  f2=cos(x)
155.  return
156.end function f2
```

這個程式中，寫作了一個叫做 InitialSettings 的函數，這個函數在 QuickWin 模式下會自動執行，它的目的是設計 QuickWin 程式的選單。

程式中沒有寫作這個函數時，Visual Fortran 會自動套用預設選單。設計選單要使用 AppendMenuQQ，使用方法如下：

integer(2) function AppendMenuQQ (menuID,flags,text, routine)

integer menuID	在第幾組選單來加入選項
integer flags	這個選項的顯示狀態，可以選擇的狀態有： $MENUGRAYED 選項無法使用，會呈現灰色狀態 $MENUDISABLED 選項無法使用，但不會呈現灰色狀態 $MENUENABLED選項可以使用 $MENUSEPARATOR 畫出一條分隔線 $MENUCHECKED在選項前打個勾 $MENUUNCHECKED 取消選項前的勾
character text	選項名稱，所傳入的字串最好記得在最末端加上字元 C，當字串設定成'string'C 時，Visual Fortran 會以 C 語言的方法來設字串。C 語言字串和 Fortran 的差別在於它的字尾會有一個結束字元。（雖然不加好像也可以用，但建議是要加比較好）
external routine	用來處理這個選項被選取時所執行的副程式，它會傳入一個 logical(kind=4)形態的參數，用來說明這個選項前面是否有「打勾」。

17-3-4 對話窗的使用

視窗程式還有一種很普遍的輸入方法，那就是出現對話窗讓使用者輸入資料。使用對話窗時，要先使用 MS Developer Studio 的資源編輯器來畫出對話窗的外觀，再寫作程式碼把這個對話窗取出來使用。

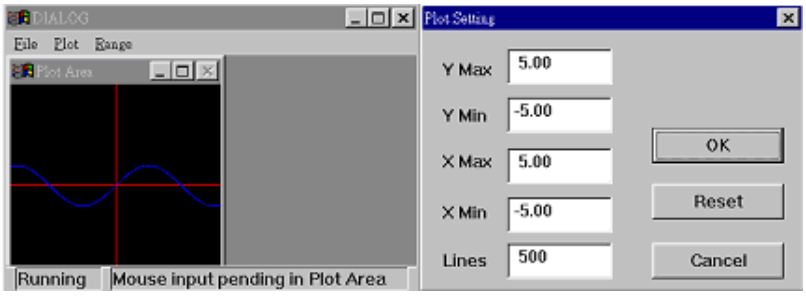

圖 **錯誤！所指定的樣式的文字不存在文件中。**-5

這個範例程式使用對話窗讓使用者設定繪圖時的 X、Y 軸範圍，以及所使用的線段數目。

DIALOG.F90

```
1.!
2.! 使用選單及對話窗的示範
3.! By Perng 1997/9/22
4.program Menu_Demo
5.use DFLIB
6.implicit none
7.  type(windowconfig) :: wc
8.  integer :: result
```

```
 9.  integer :: i,ix,iy
10.
11.  wc.numxpixels=200 ! 視窗的寬
12.  wc.numypixels=200 ! 視窗的高
13.  ! -1 代表讓程式自行去做決定
14.  wc.numtextcols=-1 ! 每行容量的文字
15.  wc.numtextrows=-1 ! 可以有幾列文字
16.  wc.numcolors=-1   ! 使用多少顏色
17.  wc.title="Plot Area"C ! 視窗的標題
18.  wc.fontsize=-1
19.  ! 根據 wc 中所定義的資料來重新設定視窗大小
20.  result=SetWindowConfig(wc)
21.  ! 把程式放入等待滑鼠訊息的狀態
22.  do while (.TRUE.)
23.    i = waitonmouseevent(MOUSE$RBUTTONDOWN, i, ix, iy)
24.  end do
25.end program
26.!
27.! 程式會自動執行這個函數, 它會設定視窗的外觀
28.!
29.logical(kind=4) function InitialSettings()
30.use DFLIB
31.implicit none
32.  logical(kind=4) :: result
33.  type(qwinfo) :: qw
34.  external PlotSin,PlotCos
35.  external SetRange
36.
37.  ! 設定整個視窗程式一開始出現的位置及大小
38.  qw.type=QWIN$SET
39.  qw.x=0
40.  qw.y=0
41.  qw.h=400
42.  qw.w=400
43.  result=SetWSizeQQ(QWIN$FRAMEWINDOW,qw)
44.
45.  ! 組織第一組選單
46.  result=AppendMenuQQ(1,$MENUENABLED,'&File'C,NUL)
47.  result=AppendMenuQQ(1,$MENUENABLED,'&Save'C,WINSAVE)
48.  result=AppendMenuQQ(1,$MENUENABLED,'&Print'C,WINPRINT)
49.  result=AppendMenuQQ(1,$MENUENABLED,'&Exit'C,WINEXIT)
50.  ! 組織第二組選單
51.  result=AppendMenuQQ(2,$MENUENABLED,'&Plot'C,NUL)
52.  result=AppendMenuQQ(2,$MENUENABLED,'&sin(x)'C,PlotSin)
53.  result=AppendMenuQQ(2,$MENUENABLED,'&cos(x)'C,PlotCos)
54.  ! 組織第三組選單
55.  result=AppendMenuQQ(3,$MENUENABLED,'&Range'C,SetRange)
56.
57.  InitialSettings=.true.
58.  return
59.end function InitialSettings
60.!
61.! 記錄全域變數
62.!
63.module Global
64.implicit none
65.  real(kind=8) :: X_Start=-5.0  ! x 軸最小範圍
66.  real(kind=8) :: X_End=5.0     ! x 軸最大範圍
67.  real(kind=8) :: Y_Top=5.0     ! y 軸最大範圍
68.  real(kind=8) :: Y_Buttom=-5.0 ! y 軸最小範圍
69.  integer     :: lines=500     ! 用多少線段來畫函數曲線
70.  integer     :: Function_Num=0 ! 使用第幾號函數來畫圖
71.end module
72.!
73.! 畫 sin 的副程式
74.!
75.subroutine PlotSin(check)
```

```
use DFLIB
use Global
implicit none
 logical(kind=4) :: check
 real(kind=8), external :: f1
 integer :: result
 check=.true.
 Function_Num=1
 ! 在第二組選單的第一個選項,也就是 sin 的前面打個勾
 result=ModifyMenuFlagsQQ(2,1,$MENUCHECKED)
 ! 把選項 cos 前的勾取消
 result=ModifyMenuFlagsQQ(2,2,$MENUUNCHECKED)
 call Draw_Func(f1)
 return
end subroutine
!
! 畫 cos 的副程式
!
subroutine PlotCos(check)
use DFLIB
use Global
implicit none
 logical(kind=4) :: check
 real(kind=8), external :: f2
 integer :: result
 check=.true.
 Function_Num=2
 ! 把選項 sin 前的勾取消
 result=ModifyMenuFlagsQQ(2,1,$MENUUNCHECKED)
 ! 在選項 cos 前打個勾
 result=ModifyMenuFlagsQQ(2,2,$MENUCHECKED)
 call Draw_Func(f2)
 return
end subroutine
!
! 按下 Range 時,會執行這個副程式
!
subroutine SetRange(check)
use Global
use Dialogm
implicit none
 logical(kind=4) :: check
 real(kind=8), external :: f1,f2
 external ReSetRange
 ! 因為想在對話窗中保留上一次的設定結果,所以安排了下列的變數
 real(kind=8),save :: OX_Start=-5.0  ! x 軸最小範圍
 real(kind=8),save :: OX_End=5.0     ! x 軸最大範圍
 real(kind=8),save :: OY_Top=5.0     ! y 軸最大範圍
 real(kind=8),save :: OY_Buttom=-5.0 ! y 軸最小範圍
 integer    ,save :: Olines=500     ! 用多少線段來畫函數曲線
 include 'resource.fd' ! 對話窗的資訊
 type(dialog) :: dl
 integer :: result    !
 character(len=20) :: str

 check=.true.
 ! 宣告要使用代碼為 IDD_INPUT 的對話窗, 並把顯示這個對話窗的資訊放
 ! 在 dl 中. 以後只要對 dl 來處理就等於對這個對話窗來工作
 result=DlgInit(IDD_INPUT, dl)

 ! 下面要對 dl 所代表的對話窗中 ID 值為 IDC_X_MIN 的欄位來設定初值
 ! 也就是設定 IDD_INPUT 中 X min 欄的內容

 ! 因為 DlgSet 無法使用 read 型態變數來設定,所以要先把它們轉換成字串
 write(str,'(f6.2)') OX_Start
 result=DlgSet(dl,IDC_X_MIN,str)
 ! 設定 X max 欄的內容
```

```
  write(str,'(f6.2)') OX_End
  result=DlgSet(dl,IDC_X_MAX,str)
  ! 設定 Y min 欄的內容
  write(str,'(f6.2)') OY_Buttom
  result=DlgSet(dl,IDC_Y_MIN,str)
  ! 設定 Y max 欄的內容
  write(str,'(f6.2)') OY_Top
  result=DlgSet(dl,IDC_Y_MAX,str)
  ! 設定 Lines 欄的內容
  write(str,'(I5)') OLines
  result=DlgSet(dl,IDC_LINES,str)
  ! 設定按下 Reset 時會執行的副程式
  result=DlgSetSub(dl,IDC_RESET, ReSetRange)
  ! 到此才真正顯示出對話窗
  result=DlgModal(dl)

  if ( result==IDOK ) then
  ! 由字串轉成數值
    result=DlgGet(dl,IDC_X_MIN,str)
    read(str,*) OX_Start
    result=DlgGet(dl,IDC_X_MAX,str)
    read(str,*) OX_End
    result=DlgGet(dl,IDC_Y_MIN,str)
    read(str,*) OY_Buttom
    result=DlgGet(dl,IDC_Y_MAX,str)
    read(str,*) OY_Top
    result=DlgGet(dl,IDC_LINES,str)
    read(str,*) OLines
  ! 設定全域變數的值, 繪圖時會取用這些數值
    X_Start=OX_Start
    X_End=OX_End
    Y_Top=OY_Top
    Y_Buttom=OY_Buttom
    Lines=OLines
  end if
  ! 由 Function_Num 的值來決定要畫出第幾個函數
  select case(Function_Num)
  case(0)
    ! Do Nothing
  case(1)
    call Draw_Func(f1)
  case(2)
    call Draw_Func(f2)
  end select

  return
end subroutine
!
! 按下 Reset 會執行這個副程式
! dlg,id,callback 會自動傳入
subroutine ReSetRange( dlg, id, callbacktype )
use DialogM
implicit none
  type(Dialog) :: dlg
  integer :: id,callbacktype
  integer :: t1,t2
  integer :: result
  include 'resource.fd'
  ! 下面這兩行沒什麼用,只是如果沒有下面兩行,Compile 時會有 Warning.
  t1=id
  t2=callbacktype
  ! 重新設定對話窗中每個欄位的內容
  result=DlgSet(dlg,IDC_X_MIN,'-5.00')
  result=DlgSet(dlg,IDC_X_MAX,' 5.00')
  result=DlgSet(dlg,IDC_Y_MIN,'-5.00')
  result=DlgSet(dlg,IDC_Y_MAX,' 5.00')
  result=DlgSet(dlg,IDC_LINES,'500')

  return
```

```
212.end subroutine
213.!
214.! 畫出所傳入的函數圖形來
215.!
216.subroutine Draw_Func(func)
217.use DFLIB
218.use Global
219.implicit none
220.  integer :: result        ! 取回繪圖函數運作狀態
221.  integer(kind=2) :: color  ! 設定顏色用
222.  real(kind=8) :: step     ! 迴圈的增量
223.  real(kind=8) :: x,y      ! 繪圖時使用,每條小線段都連接
224.  real(kind=8) :: NewX,NewY ! (x,y)及(NewX,NewY)
225.  real(kind=8), external :: func ! 待繪圖的函數
226.  type(wxycoord) :: wt     ! 傳回上一次的虛擬座標位置
227.
228.  call ClearScreen($GCLEARSCREEN) ! 清除螢幕
229.  ! 設定虛擬座標範圍大小
230.  result=SetWindow( .true. , X_Start, Y_Top, X_End, Y_Buttom )
231.
232.  ! 使用全彩 RGB 0-255 的 256 種色階來設定顏色
233.  color=RGBToInteger(255,0,0)      ! 把控制 RGB 的三個值濃縮到 color 中
234.  result=SetColorRGB(color)        ! 利用 color 來設定顏色
235.
236.  call MoveTo_W(X_Start,0.0_8,wt)   ! 畫 X 軸
237.  result=LineTo_W(X_End,0.0_8)     !
238.  call MoveTo_W(0.0_8,Y_Top,wt)     ! 畫 Y 軸
239.  result=LineTo_W(0.0_8,Y_Buttom)   !
240.
241.  step=(X_End-X_Start)/lines       ! 計算小線段間的 X 間距
242.  ! 參數#FF0000 是使用 16 進位的方法來表示一個整數
243.  result=SetColorRGB(#FF0000)
244.
245.  ! 開始繪製小線段
246.  do x=X_Start,X_End-step,step
247.    y=func(x)      ! 線段的左端點
248.    NewX=x+step
249.    NewY=func(NewX) ! 線段的右端點
250.    call MoveTo_W(x,y,wt)
251.    result=LineTo_W(NewX,NewY)
252.  end do
253.
254.  ! 設定程式結束後,視窗會繼續保留
255.  result=SetExitQQ(QWIN$EXITPERSIST)
256.end subroutine Draw_Func
257.!
258.! 所要繪圖的函數
259.!
260.real(kind=8) function f1(x)
261.implicit none
262.  real(kind=8) :: x
263.  f1=sin(x)
264.  return
265.end function f1
266.
267.real(kind=8) function f2(x)
268.implicit none
269.  real(kind=8) :: x
270.  f2=cos(x)
271.  return
272.end function f2
```

這個程式的寫作過程有必要做一個介紹。

1. 開啟 QuickWin 模式的 Project，把 DIALOG.F90 加入 Project 中。

2. 按下 Insert/Resource 選項，選擇要新增 Dialog。

3. 利用 Dialog Resource 編輯工具來設計出所要使用的 Dialog

4. 儲存編輯好的對話窗，並把*.RC 檔加入 Project 中

這 4 個步驟都不會太難，只有第 3 步需要更進一步說明。新增一個 Dialog 型態的 Resource 後，可以在螢幕上看到一個剛出生的對話窗，它只有 OK、Cancel 這兩個按鈕而已，螢幕上同時還會出現編輯工具，可以經由滑鼠把這些小元件拉到對話窗上來編輯。這些小元件中有一些資料需要設定，例如元件的 ID 值等等。

製作對話窗的詳細步驟如下：

圖 **錯誤！所指定的樣式的文字不存在文件中。**-6

選取 Insert 中的 Rscource 選項

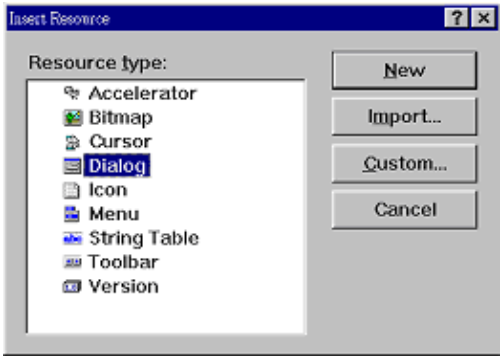

圖 **錯誤！所指定的樣式的文字不存在文件中。**-7

選擇所要新增的 Resource 型態是 Dialog（對話窗）。

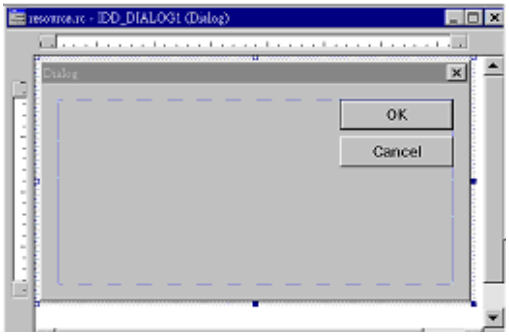

圖 **錯誤！所指定的樣式的文字不存在文件中。**-8

會出現一個基本的對話窗來供編輯、修改

 圖 **錯誤! 所指定的樣式的文字不存在文件中。**-9

這個視窗是用來編輯對話窗的元件，可以用滑鼠來拖曳這些元件。

圖 **錯誤! 所指定的樣式的文字不存在文件中。**-10

把所需要的元件用滑鼠拖曳到對話窗中。

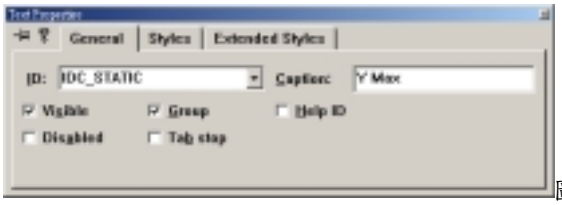

圖 **錯誤! 所指定的樣式的文字不存在文件中。**-11

在 static 元件上按兩下，會出現屬性設定視窗，在 Caption 欄可以改變顯示文字。ID 欄的值用來代表這個元件，在程式中可以用 ID 欄所指定的代號來讀寫元件的內容。

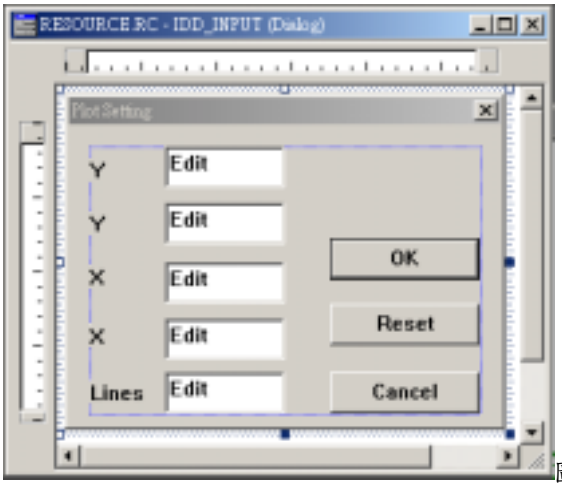

圖 **錯誤! 所指定的樣式的文字不存在文件中。**-12

這是編輯好的對話窗，設計對話窗時除了要注意外觀外，最重要的是設定每個元件的 ID 代號。因為程式會從 ID 代號來讀取對話窗中所輸入的資料。

製作好對話窗之後，會儲存成*.RC 檔，請記得把*.RC 檔加入 Project 中，不然對話窗做好了也等於沒做。在儲存 RC 檔時，Visual Fortran 會自動產生一個*.FD 檔，這個檔案的內容是 FORTRAN 程式碼，它會宣告一些跟對話窗相關的變數，這些變數可以用來操作對話窗，它們是元件的代碼，讀者可以試著把*.FD 檔拿出來看。寫作具備對話窗功能的程式時，要使用 Visual Fortran 所內附的 MODULE DialogM，還要 INCLUDE '*.FD'，把產生出來的*.FD 檔插入程式碼中。

跟上一個範例程式比較起來，這裏只新增了兩個副程式來控制對話窗。其它的部分幾乎大同小異，只有把上個程式中的某些局部變數改成全域變數。

subroutine SetRange　負責顯示對話窗。
subroutine ReSet　　負責處理對話窗中 Reset 鈕被按下的訊息

程式碼中已經有相當程度的說明，在此只對幾個函式做介紹：

logical(4) function DLGINIT (id, dlg)

這個函數的目的是做顯示對話窗前的預備工作，id 值是指在製作對話窗時所給定的對話窗代碼。可以在*.FD 檔中找到所定義的數值及變數。dlg 會傳回對話窗的資訊。

logical(4) function DLGSET (dlg, controlid, value [, index])

設定對話窗上各個欄位的內容。

`type(Dialog) :: Dlg`	變數要先經過 `DlgInit` 處理。在此它可以用來代表要對某個對話窗來工作
`integer controlid`	在製作對話窗時元件的 `ID` 值。
`real/integer/character value`	`value` 的內容會被設定到視窗中 `controlid` 值所指的元件上。
`integer index`	這個值不一定需要，是元件中又具備有小元件時用來指定小元件用的。

logical(4) function　DLGSETSUB (dlg, id, sub [, index])

設定某個元件被按下後會執行副程式 sub。

`type(dialog) :: dlg`	變數要先經過 `DlgInit` 處理，在此它可以用來代表要對某個對話窗來工作
`integer :: id`	在製作對話窗時元件的 `ID` 值。

`external sub`	元件按下時會執行 `sub`，副程式 `sub` 會得到 3 個參數：
	`type(dialog) :: dlg`
	表示現在所使用的對話窗
	`integer :: id`
	被按下的元件代碼
	`integer :: callbacktype`
	這個值可以不去理會

`integer index`	這個值不一定需要，是一個元件當中又具備有小元件時用來指定小元件用的。

對話窗的元件很豐富，建議讀者可以不用每一項功能都去熟悉。其實只要學會上面那幾個元件就可以處理很多問題了。

下面的範例程式當使用者按下 Setcolor 選項時，會出現一個對話窗來讓使用者拉動捲軸來設定 R、G、B 色光的強度，改變設定的同時，會以三色光混合成的顏色畫出一個實心方形。讀者還可以使用 File 中的 Save 來儲存目前的設定。

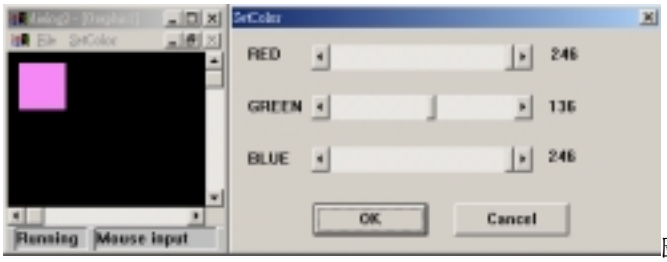

圖 **錯誤! 所指定的樣式的文字不存在文件中。**-13

SETCOLOR.F90

```
 1.!
 2.! 對話窗的示範之二
 3.! By Perng 1997/09/26
 4.program main
 5.use DFLIB
 6.implicit none
 7.  integer :: i,ix,iy
 8.
 9.  call ClearScreen($GCLEARSCREEN)
10.  do while(.true.)
11.    i = waitonmouseevent(MOUSE$LBUTTONDOWN, i, ix, iy)
12.  end do
13.end program
14.!
15.! 記錄全域變數
16.!
17.module Global
18.implicit none
19.  integer, save :: red=0
20.  integer, save :: green=0
21.  integer, save :: blue=0
22.end module
23.!
24.! 自訂選單
25.!
26.logical(kind=4) function InitialSettings()
27.use DFLIB
28.implicit none
29.  logical(kind=4) result
30.  external FileOpen,FileSave
31.  external SetColorDialog
32.
33.  result=AppendMenuQQ(1,$MENUENABLED,'File'C,NUL)
34.  result=AppendMenuQQ(1,$MENUENABLED,'&Open'C,FileOpen)
35.  result=AppendMenuQQ(1,$MENUENABLED,'&Save'C,FileSave)
36.  result=AppendMenuQQ(1,$MENUENABLED,'&Exit'C,WINEXIT)
37.
38.  result=AppendMenuQQ(2,$MENUENABLED,'&SetColor',SetColorDialog)
39.
40.  InitialSettings=result
41.  return
42.end function
43.!
```

```
!處理 Open
!
subroutine FileOpen(check)
use Global
implicit none
  logical :: check
  check=.true.
  ! 開啟一個空白字元的檔案, 會出現檔案選擇的對話窗來給使用者選擇檔案
  open(unit=10, file=' ')
  read(10,*) red
  read(10,*) green
  read(10,*) blue
  close(10)
  call DrawObject()
  return
end subroutine
!
!處理 Save
!
subroutine FileSave(check)
use Global
implicit none
  logical :: check
  check=.true.
  ! 使用檔案選擇的對話窗
  open(unit=20, file=' ')
  write(20,*) red
  write(20,*) green
  write(20,*) blue
  close(20)
  return
end subroutine
!
! 按下 SetColor 時會執行這個函式
!
subroutine SetColorDialog(check)
use Global
use DFLIB
use dialogm
implicit none
  logical :: check
  integer :: result
  include 'resource.fd'
  type(dialog) :: dlg
  external ShowColor
  character(len=10) output

  check=.true.
  result=DlgInit(IDD_SETCOLOR, dlg) ! 初始對話窗
  ! 設定捲軸可以捲動的範圍
  result=DlgSet( dlg, IDC_SCROLLBAR1, 256, dlg_range )
  result=DlgSet( dlg, IDC_SCROLLBAR2, 256, dlg_range )
  result=DlgSet( dlg, IDC_SCROLLBAR3, 256, dlg_range )
  ! 設定捲軸目前的位置
  result=DlgSet( dlg, IDC_SCROLLBAR1, red, dlg_position )
  result=DlgSet( dlg, IDC_SCROLLBAR2, green, dlg_position )
  result=DlgSet( dlg, IDC_SCROLLBAR3, blue, dlg_position )
! 上面有比較奇怪的用法, dlg_range 代表要設定捲軸元件的範圍值
! dlg_position 代表要設定捲軸元件的位置. 讀者可以想像當型態為 dialog
! 的 dlg 被宣告時, 編譯器會自動宣告出 dlg_range, dlg_position 等等
! 的變數並設定好它們的數值.

  ! 寫出紅,綠,藍三色光的強度數值
  write(output,"(I3)") red
  result=DlgSet( dlg, IDC_VALUE_RED, output )
  write(output,"(I3)") green
  result=DlgSet( dlg, IDC_VALUE_GREEN, output )
  write(output,"(I3)") blue
  result=DlgSet( dlg, IDC_VALUE_BLUE, output )
```

```
113.  ! 設定拉動捲軸時會執行的函式
114.  result=DlgSetSub( dlg, IDC_SCROLLBAR1, ShowColor )
115.  result=DlgSetSub( dlg, IDC_SCROLLBAR2, ShowColor )
116.  result=DlgSetSub( dlg, IDC_SCROLLBAR3, ShowColor )
117.  ! 顯示對話窗
118.  result=DlgModal(dlg)
119.
120.  return
121.end subroutine
122.!
123.! 取出三色光的設定並畫出一個實心方形
124.!
125.subroutine ShowColor(dlg,id,callbacktype)
126.use Global
127.use DFLIB
128.use dialogm
129.implicit none
130.  type(dialog) :: dlg
131.  integer :: id, callbacktype
132.  integer :: result
133.  character(len=10) :: output
134.  include 'resource.fd'
135.  ! 取出捲軸的位置
136.  result=DlgGet( dlg, IDC_SCROLLBAR1, red, dlg_position )
137.  result=DlgGet( dlg, IDC_SCROLLBAR2, green, dlg_position )
138.  result=DlgGet( dlg, IDC_SCROLLBAR3, blue, dlg_position )
139.  ! 因為顏色的變化範圍是 0-255, 而捲軸的範圍是 1-256, 所以要減 1
140.  red=red-1
141.  green=green-1
142.  blue=blue-1
143.
144.  select case(id)
145.  case(IDC_SCROLLBAR1)  ! 第一個捲軸設定紅色光強度
146.    write(output,"(I3)") red
147.    result=DlgSet( dlg, IDC_VALUE_RED, output )
148.  case(IDC_SCROLLBAR2)  ! 第二個捲軸設定綠色光強度
149.    write(output,"(I3)") green
150.    result=DlgSet( dlg, IDC_VALUE_GREEN, output )
151.  case(IDC_SCROLLBAR3)  ! 第三個捲軸設定藍色光強度
152.    write(output,"(I3)") blue
153.    result=DlgSet( dlg, IDC_VALUE_BLUE, output )
154.  end select
155.
156.  call DrawObject()
157.
158.  return
159.end subroutine
160.!
161.! 以設定的顏色畫出實心方形
162.!
163.subroutine DrawObject()
164.use Global
165.use DFLIB
166.implicit none
167.  integer :: result
168.  integer :: color
169.
170.  color=RGBToInteger(red,green,blue)
171.  result=SetColorRGB(color)
172.  result=Rectangle($GFILLINTERIOR,10,10,50,50)
173.
174.  return
175.end subroutine
```

程式碼中已經有足夠的註解，在此不再多做說明。

17-4 Visual Fortran 繪圖函式總覽

Visual Fortran 的繪圖函式大部分是函數，呼叫後的傳回值大多是用來說明函數是否正常執行。幾何繪圖函式通常有兩個版本，函式名稱最後為「_W」的使用虛擬座標，不然就使用視窗座標。使用虛擬座標時，都使用雙精確度浮點數來傳遞座標值。

在下面的函式說明中，不會詳細列出兩種版本的繪圖函式。在函式名稱最後有[_W]代表這個函式另外還有使用虛擬座標的版本。例如 ARC[_W]代表有兩個函式，分別叫做 ARC 及 ARC_W，ARC_W 使用虛擬座標，參數型態會改用雙精確度浮點數。

integer(2) function ARC[_W](x1,y1,x2,y2,x3,y3,x4,y4)

畫弧形，要指定一個矩形範圍，弧形會畫在這個矩形範圍內的橢圖上

integer(2) x1,y1,x2,y2	矩形範圍的兩個端點
integer(2) x3,y3	弧形的起始向量
integer(2) x4,y4	弧形的結束向量

subroutine CLEARSCREEN(area)

清除螢幕

integer(4) area	設定所要清除的範圍，有下列的定義值可以使用： $GCLEARSCREEN 清除整個螢幕 $GVIEWPORT 清除目前所設定的可用範圍 $GWINDOW 清除所設定的文字視窗範圍

integer(2) function ELLIPSE[_W](control, x1, y1, x2, y2)

在指定的矩形範圍內畫橢圓

integer(2) control	設定要填滿內部或是只畫外框，有下列的定義值可以使用： $GFILLINTERIOR 塗滿整個內部 $GBORDER 只畫外框
integer(2) x1,y1	矩形端點 1
integer(2) x2,y2	矩形端點 2

integer(2) function FLOODFILL[_W](x,y,bcolor)

把一個封閉空間上色。

integer(4) x,y	封閉區間內的座標點
integer bcolor	用 INDEX 模式來設定顏色

integer(2) function FLOODFILLRGB[_W](x,y,color)

把一個封閉空間上色。

integer(4) x,y	封閉區間內的座標點
integer color	用 RGB 模式來設定顏色

integer(2) function GETCURRENTPOSITION[_W](t)

查詢目前的畫筆位置

type(xycoord) t	傳回目前畫筆的所在位置，使用虛擬座標時 t 的型態為 type(wxycoord)

integer(2) function GETPIXEL[_W](x, y)

取得(x,y)點的顏色，傳回值為 INDEX 模式的顏色值。

integer(4) function GETPIXELRGB[_W](x, y)

取得(x,y)點的顏色，傳回值為 RGB 模式的顏色值。

subroutine GETPIXELS(n, x, y, color)

一次取得許多點的顏色資料（Index Color 模式）

integer(4) n	想取得多少點
integer(2) x(n),y(n)	要傳入兩個陣列，用來設定所要讀取的座標。
integer(2) color(n)	顏色資料會放在這個陣列中

subroutine GETPIXELSRGB(n, x, y, color)

一次取得許多點的顏色資料(RGB Color 模式)，參數同上，不過 color 改用 integer(4) 型態。

subroutine INTEGERTORGB(rgb, red, green, blue)

把傳入的 rgb 整數值分解成 red、green、blue 三部分

integer(4) rgb	傳入所要分解的 rgb 顏色值
integer(4) red,green,blue	傳回分解出來的紅、綠、藍色光值

integer(2) function LINETO[_W](x, y)

從目前畫筆位置到(x,y)間畫一條直線。

integer(2) x,y	指定畫線的終點

subroutine MOVETO[_W](x, y, t)

integer(2) x,y	設定畫筆座標位置
type(xycoord) t	傳回上次的畫筆位置

integer(2) function PIE[_W](fill,x1,y1, x2,y2, x3,y3, x4,y4)

畫扇形，要指定一個矩形範圍，扇形會畫在這個矩形範圍內的橢圖上

integer(2) fill	設定只畫外框或畫實心的扇形
integer(2) x1,y1,x2,y2	橢圓的矩形範圍
integer(2) x3,y3,x4,y4	扇形的起始及終點向量

integer(2) function POLYGON[_W](control, ppoints, cpoints)

繪製多邊形

integer(2) control	設定只畫外框或畫實心的多邊形
type(xycoord) ppints(cpoints)	用陣列來傳入多邊形的頂點座標
integer(2) cpoints	傳入的座標點數目

integer(2) function RECTANGLE[_W](control, x1, y1, x2, y2)

畫矩形

integer(2) control	設定只畫外框或畫實心的矩形
integer(2) x1,y1,x2,y2	設定矩形的範圍

integer(4) function RGBTOINTEGER(red, green, blue)

把傳入的 red,green,blue 值濃縮成一個長整數傳回。red 值會放在 0~7 bits，green 會放在 8~15 bits，blue 會放在 16~23 bits。

integer(2) red,green,blue	傳入的紅、綠、藍色光值

integer(2) function SETPIXEL[_W](x,y)

用目前設定的顏色在指定座標處畫點（Index Color 模式）

integer(2) x,y	畫點的座標

integer(2) function SETPIXELRGB[_W](x,y)

用目前設定的顏色在指定座標處畫點（RGB Color 模式）

integer(2) x,y	畫點的座標

subroutine SETPIXELS(n, x, y, color)

一次畫許多個點（Index Color 模式）

integer(2) n	要畫幾個點
integer(2) x(n),y(n)	傳入要畫的座標點
integer(2) color(n)	傳入每個點的顏色

subroutine SETPIXELSRGB(n, x, y, color)

一次畫許多個點（RGB Color 模式），參數同上，除了 color 改用 integer(4)型態。

下面要介紹跟文字相關的函式，使用 WRITE/READ 來讀寫 QuickWin 視窗時可以使用下列的函式。

integer(2) function DISPLAYCURSOR(toggle)

控制游標的顯示

integer(2) toggle	控制是否顯示游標，有下列的定義值可以使用 $GCURSOROFF 不顯示游標 $GCURSORON 顯示游標

integer(4) function GETBKCOLOR()

傳回背景顏色（Index Color 模式）

integer(4) function GETBKCOLORRGB()

傳回背景顏色（RGB Color 模式）

integer(4) function GETTEXTCOLOR()

傳回目前文字的顏色（Index Color 模式）

integer(4) function GETTEXTCOLORRGB()

傳回目前文字的顏色（RGB Color 模式）

subrotuine GETTEXTPOSITION(t)

傳回目前文字的輸出位置

type(rccord) t	傳回目前文字的輸出位置

subroutine GETTEXTWINDOW(r1, c1, r2, c2)

傳回目前文字視窗的設定範圍

integer(2) r1,c1,r2,c2	傳回目前文字視窗的設定範圍

subroutine OUTTEXT(text)

在目前的文字輸出位置寫出字串 text 的內容

character*(*) text	所要輸出的字串內容

subroutine SCROLLTEXTWINDOW(rows)

把文字向上捲動 rows 行

integer(2) rows	想要捲動的行數

subroutine SETBKCOLOR(color)

設定背景顏色（Index Color 模式）

integer(4) color	所要設定的顏色

subroutine SETBKCOLORRGB(color)

設定背景顏色（RGB Color 模式）

integer(4) color	所要設定的顏色

subroutine SETTEXTPOSITION(row, column, t)

設定文字輸出的位置

integer(2) row,column	設定文字輸出的位置

subroutine SETTEXTWINDOW(r1, c1, r2, c2)

設定文字視窗的範圍

integer(2) r1,c1,r2,c2	設定文字視窗的左上角及右下角範圍

integer(2) function WRAPON(option)

設定文字輸出超過視窗範圍時是否會自動斷行

integer(2) option	有兩個數值可代入 $GWRAPOFF 會自動斷行 $GWRAPON 不會斷行

下面是使用字型的函式，這裏的函式可以使用 Windows 中所安裝的 TrueType 字型。

integer(2) function GETFONTINFO(font)

取得目前使用字型的資訊

type(fontinfo) font	傳回目前使用字型的資訊

integer(2) function GETGTEXTEXTENT(text)

傳回這個字串輸出時會佔用的螢幕點數

character*(*) text	傳入的字串

integer(2) function GETGTEXTROTATION()

傳回目前設定的字串輸出角度

integer(4) result	傳回輸出角度值的 10 倍

subroutine SETGTEXTROTATION(degrees)

設定字串的輸出角度

integer(4) degrees	用角度值的 10 倍數值來設定文字的輸出角度

subroutine OUTGTEXT(text)

在目前畫筆位置輸出字串

character*(*) text	所要輸出的字串

integer(2) function INITIALIZEFONTS()

使用字型前的準備工作，要使用 TrueType 字型時，一定要先呼叫這個函式。

integer(2) function SETFONT(options)

設定所要使用的字型

character*(*) options	使用文字敘述來設定要使用的字型，要描述的內容有「字型」、「長寬」、「樣式」等等。

option 的設定方法如下：

t'FontName' 指定使用的字型
hy 其中的 y 值會設定字型的高度
wx 其中的 x 值會設定字型的寬度
f 使用固定大小的字型
p 使用可變大小的字型
v 使用向量字型
r 使用點矩陣字型
e 設定為粗體的樣式
u 設定為有底線的樣式
i 設定為斜體的樣式
b 自動設定沒有指定的資料
n 用編號來設定字型

舉一個例子，result=SetFont('t''Times New Roman''h12w10')會使用 12*10 的 Times New Roman 字型。

接下來要介紹跟影像相關的函式

subroutine GETIMAGE[_W](x1,y1,x2,y2,image)

抓取螢幕上一塊矩形區域影像到記憶體中

integer(2) x1,y1,x2,y2	指定一個矩形區域
integer(1) image(n)	用來儲存圖形的陣列

integer(2) IMAGESIZE[_W](x1,y1,x2,y2)

計算抓取這個矩形區域影像所要使用的陣列大小

integer(2) x1,y1,x2,y2	指定所要計算的矩形區域

subroutine PUTIMAGE[_W](x,y,image,action)

把圖形貼到螢幕上

integer(2) x,y	指定要貼到螢幕上的位置，圖形左上角會對應到這個位置
integer(1) image(*)	所要顯示的影像資料
integer(2) action	控制圖形要貼到螢幕上前所要做的計算，有下列的定義可代入： $GAND 和原先在螢幕上的點做 AND 運算 $GOR 和原先在螢幕上的點做 OR 運算 $GXOR 和原先在螢幕上的點做 XOR 運算

integer(2) LOADIMAGE[_W](filename, x, y)

讀取 bmp 圖檔，把它顯示在畫面上。

character*(*) filename	指定要讀入的圖檔
integer(4) x, y	顯示圖形時所使用的左上角座標

integer(2) function SAVEIMAGE[_W](filename, x1,y1, x2,y2)

把螢幕上的一塊矩形區域影像儲存成 bmp 圖檔

character*(*) filename	所要儲存的檔案
integer(4) x1,y1,x2,y2	指定矩形的範圍