# W I N 3 2 − F O R T R A N

by Angelo Graziosi

## I N T R O D U C T I O N

A basic question for a fortranner is: How to create Fortran applications with GUI interface? More advanced Fortran GUI programs could be created with GTK−Fortran library (https://github.com/jerryd/gtk−fortran), i.e. using the interoperability between C and Fortran, which comes with the Fortran 2003 standard.

Following that example, we have created modules which partially interface BGI (Borland Graphics Interface). They have been described elsewhere on this WEB site.

On Windows we can have Fortran GUI programs using an interface to Windows itself. This is what we present in the following: partial interface to Windows which allow for creating simple Windows applications in Fortran.

Rudimentary modules which implement this are contained in win32.f90, win32boxes.f90 and win32app.f90 source files. The first contains the interface itself, the second tries to recover an old idea we implemented creating a dialog C++ library with the old Borland C++ 2.0 compiler (around 1991). The third, tries to do things in World Coordinate System.

A few examples of these applications are attached below. As always, details in the comments.

A special thanks goes to T. Burnus, F−X. Coudert and J. Blomqvist for their valuable suggestions.

---

This document has been created using EMACS (and some "friends" tools like ps2pdf, pdftk etc..).

```
!
! (Partial) Fortran Interface to the Windows API Library
! by Angelo Graziosi  (firstname.lastnameATalice.it)
! Copyright Angelo Graziosi
!
! It is distributed in the hope that it will be useful,
! but WITHOUT ANY WARRANTY; without even the implied warranty of
! MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
!
!
! DESCRIPTION
!   This is the 'win32' module.
!   Just to start with Windows Fortran Applications...
!
!   An idea from: http://home.comcast.net/~kmbtib/Fortran_stuff/HelloWin2.f90
!
! NOTE
!   For Microsoft, the type "long" is always (in Windows 32 and 64 systems)
!   a 32 bit integer.
!   For GNU/Linux it is a 32/64 bit integer for systems 32/64 rispectively.
!   So we have to adopt: C_LONG --> C_INT, being C_INT a 32 bit integer
!   in any case.
!
!   See the thread: http://gcc.gnu.org/ml/fortran/2013-07/msg00087.html
!   See also: http://cygwin.com/cygwin-ug-net/programming.html#gcc-64
!
! BTW
!   The usage of iany() (Fortran 2008) need of GCC >= 4.7
!
!   Notice that:
!
!       int(0,UINT_T)    -->  0_UINT_T
!       int(0,WPARAM_T)  -->  0_WPARAM_T
!       int(0,LPARAM_T)  -->  0_LPARAM_T
!       ...
!

module win32
  use, intrinsic :: iso_c_binding, only: C_CHAR, c_f_pointer, C_FUNPTR, C_INT, &
       C_INT8_T, C_INTPTR_T, C_LONG, C_NEW_LINE, C_NULL_CHAR, C_NULL_PTR, &
       C_PTR, C_SHORT
  implicit none
  private

  ! Common useful constants
  integer, parameter, public :: MAX_LEN = 256
  integer, parameter, public :: MAX_FMT = 12

  ! =================
  !   WIN32 ALIASES
  ! =================

  ! Using directly C_INT to define DWORD_T and LONG_T maybe misleading,
  ! so we adopt the Tobias tips
  ! (http://gcc.gnu.org/ml/fortran/2013-07/msg00090.html).
  !
  integer, parameter :: C_MS_LONG = C_INT

  integer, parameter, public :: BYTE_T = C_INT8_T
  integer, parameter, public :: DWORD_T = C_MS_LONG
  integer, parameter, public :: HANDLE_T = C_INTPTR_T
  integer, parameter, public :: INT_T = C_INT
  integer, parameter, public :: INT_PTR_T = C_INTPTR_T
  integer, parameter, public :: LONG_T = C_MS_LONG
  integer, parameter, public :: LONG_PTR_T = C_INTPTR_T
  integer, parameter, public :: SHORT_T = C_SHORT
  integer, parameter, public :: UINT_PTR_T = C_INTPTR_T
  integer, parameter, public :: WORD_T = C_SHORT

  integer, parameter, public :: ATOM_T = WORD_T
  integer, parameter, public :: BOOL_T = INT_T
  integer, parameter, public :: COLORREF_T = DWORD_T
  integer, parameter, public :: HBITMAP_T = HANDLE_T
  integer, parameter, public :: HBRUSH_T = HANDLE_T
  integer, parameter, public :: HCURSOR_T = HANDLE_T
  integer, parameter, public :: HDC_T = HANDLE_T
  integer, parameter, public :: HGDIOBJ_T = HANDLE_T
  integer, parameter, public :: HICON_T = HANDLE_T
```

```
integer, parameter, public :: HINSTANCE_T = HANDLE_T
integer, parameter, public :: HMENU_T = HANDLE_T
integer, parameter, public :: HMODULE_T = HINSTANCE_T
integer, parameter, public :: HMONITOR_T = HANDLE_T
integer, parameter, public :: HPEN_T = HANDLE_T
integer, parameter, public :: HWND_T = HANDLE_T
integer, parameter, public :: LPARAM_T = LONG_PTR_T
integer, parameter, public :: LRESULT_T = LONG_PTR_T
integer, parameter, public :: UINT_T = INT_T
integer, parameter, public :: WPARAM_T = UINT_PTR_T

! ===============
!   WIN32 TYPES
! ===============

type, public, bind(C) :: WNDCLASSEX_T
   integer(UINT_T) :: cbSize
   integer(UINT_T) :: style
   type(C_FUNPTR) :: lpfnWndProc ! WNDPROC
   integer(INT_T) :: cbClsExtra
   integer(INT_T) :: cbWndExtra
   integer(HINSTANCE_T) :: hInstance
   integer(HICON_T) :: hIcon
   integer(HCURSOR_T) :: hCursor
   integer(HBRUSH_T) :: hbrBackground
   type(C_PTR) :: lpszMenuName  ! LPCTSTR
   type(C_PTR) :: lpszClassName ! LPCTSTR
   integer(HICON_T) :: hIconSm
end type WNDCLASSEX_T

type, public, bind(C) :: POINT_T
   integer(LONG_T) :: x
   integer(LONG_T) :: y
end type POINT_T

type, public, bind(C) :: MSG_T
   integer(HWND_T) :: hWnd
   integer(UINT_T) :: message
   integer(WPARAM_T) :: wParam
   integer(LPARAM_T) :: lParam
   integer(DWORD_T) :: time
   type(POINT_T) :: pt
end type MSG_T

type, public, bind(C) :: RECT_T
   integer(LONG_T) :: left
   integer(LONG_T) :: top
   integer(LONG_T) :: right
   integer(LONG_T) :: bottom
end type RECT_T

type, public, bind(C) :: PAINTSTRUCT_T
   integer(HDC_T) :: hdc
   integer(BOOL_T) :: fErase
   type(RECT_T) :: rcPaint
   integer(BOOL_T) :: fRestore
   integer(BOOL_T) :: fIncUpdate
   integer(BYTE_T) :: rgbReserved(32)
end type PAINTSTRUCT_T

type, public, bind(C) :: MONITORINFO_T
   integer(DWORD_T) :: cbSize
   type(RECT_T) :: rcMonitor
   type(RECT_T) :: rcWork
   integer(DWORD_T) :: dwFlags
end type MONITORINFO_T

! ================================
!   WIN32 CONSTANTS AND VARIABLES
! ================================

! An alternative to the function null_p()
character(C_CHAR), pointer, public :: NULL_LPSTR(:) => null()

type(RECT_T), pointer, public :: NULL_RECT_T => null()

integer(HANDLE_T), parameter, public :: NULL_T = 0
integer(BOOL_T), parameter, public :: FALSE_T = 0
```

```
integer(BOOL_T), parameter, public :: TRUE_T = 1

type(C_PTR), parameter, public :: NULL_PTR_T = C_NULL_PTR

! C string constants alaises using the ASCII name.
character(C_CHAR), parameter, public :: NUL = C_NULL_CHAR
character(C_CHAR), parameter, public :: NL = C_NEW_LINE

! COLORREF (Z'00BBGGRR') constants
integer(COLORREF_T), parameter, public :: BLACK_COLOR = 0        ! Z'00000000'
integer(COLORREF_T), parameter, public :: CYAN_COLOR = 16776960  ! Z'00FFFF00'
integer(COLORREF_T), parameter, public :: YELLOW_COLOR = 65535   ! Z'0000FFFF'
integer(COLORREF_T), parameter, public :: WHITE_COLOR = 16777215 ! Z'00FFFFFF'

! Device−specific information index (/usr/include/w32api/wingdi.h)
integer(INT_T), parameter, public :: ASPECTX = 40
integer(INT_T), parameter, public :: ASPECTY = 42
integer(INT_T), parameter, public :: ASPECTXY = 44

! Window default position and/or dimension.
! The C/C++ definition is ((int)0x80000000), i.e. (int)2147483648.
! Given the range of int (4 bytes) is [−2147483648,2147483647], 2147483648
! means −2147483648. The right way to obtain this is as follows:
! One cannot use CW_USEDEFAULT = −2147483648, because it would use an
! unary minus operator on the integer constant (+)2147483648, which
! does not exist! (the maximum is 2147483647!)
! See also this explanation
! http://gcc.gnu.org/ml/fortran/2013−12/msg00083.html,
! and the relative thread, for a similar question.
integer(INT_T), parameter, public :: CW_USEDEFAULT = −2147483647−1

! Class styles (/usr/include/w32api/winuser.h)
integer(UINT_T), parameter, public :: CS_VREDRAW = 1     ! Z'00000001'
integer(UINT_T), parameter, public :: CS_HREDRAW = 2     ! Z'00000002'
integer(UINT_T), parameter, public :: CS_SAVEBITS = 2048 ! Z'00000800'

! DrawText formats (/usr/include/w32api/winuser.h)
integer(UINT_T), parameter, public :: DT_CENTER = 1
integer(UINT_T), parameter, public :: DT_VCENTER = 4
integer(UINT_T), parameter, public :: DT_SINGLELINE = 32

! Hatch style of the brush (/usr/include/w32api/wingdi.h)
integer(INT_T), parameter, public :: HS_DIAGCROSS = 5

! IDC_* definitions for make_int_resource() (/usr/include/w32api/winuser.h)
integer(WORD_T), parameter, public :: IDC_ARROW = 32512
integer(WORD_T), parameter, public :: IDC_CROSS = 32515
integer(WORD_T), parameter, public :: IDC_HAND = 32649
integer(WORD_T), parameter, public :: IDC_WAIT = 32514

! IDI_* definitions for make_int_resource() (/usr/include/w32api/winuser.h)
integer(WORD_T), parameter, public :: IDI_APPLICATION = 32512
integer(WORD_T), parameter, public :: IDI_ASTERISK = 32516
integer(WORD_T), parameter, public :: IDI_ERROR = 32513
integer(WORD_T), parameter, public :: IDI_EXCLAMATION = 32515
integer(WORD_T), parameter, public :: IDI_HAND = 32513
integer(WORD_T), parameter, public :: IDI_INFORMATION = 32516
integer(WORD_T), parameter, public :: IDI_QUESTION = 32514
integer(WORD_T), parameter, public :: IDI_WARNING = 32515
integer(WORD_T), parameter, public :: IDI_WINLOGO = 32517

! MessageBox() buttons and return values (/usr/include/w32api/winuser.h)
integer(UINT_T), parameter, public :: MB_ICONASTERISK = 64    ! Z'00000040'
integer(UINT_T), parameter, public :: MB_ICONHAND = 16        ! Z'00000010'
integer(UINT_T), parameter, public :: MB_ICONERROR = MB_ICONHAND
integer(UINT_T), parameter, public :: MB_ICONEXCLAMATION = 48 ! Z'00000030'
integer(UINT_T), parameter, public :: MB_ICONINFORMATION = MB_ICONASTERISK
integer(UINT_T), parameter, public :: MB_ICONQUESTION = 32    ! Z'00000020'
integer(UINT_T), parameter, public :: MB_OK = 0               ! Z'00000000'
integer(UINT_T), parameter, public :: MB_YESNO = 4            ! Z'00000004'
integer(UINT_T), parameter, public :: MB_YESNOCANCEL = 3      ! Z'00000003'
!
integer(INT_T), parameter, public :: IDCANCEL = 2
integer(INT_T), parameter, public :: IDOK = 1
integer(INT_T), parameter, public :: IDYES = 6

! Specifies how messages are to be handled (/usr/include/w32api/winuser.h)
integer(UINT_T), parameter, public :: PM_NOREMOVE = 0  ! Z'00000000'
```

```
integer(UINT_T), parameter, public :: PM_REMOVE = 1    ! Z'00000001'
integer(UINT_T), parameter, public :: PM_NOYIELD = 2   ! Z'00000002'

! Pen styles (/usr/include/w32api/wingdi.h)
integer(INT_T), parameter, public :: PS_SOLID = 0
integer(INT_T), parameter, public :: PS_DASH = 1
integer(INT_T), parameter, public :: PS_DOT = 2
integer(INT_T), parameter, public :: PS_DASHDOT = 3
integer(INT_T), parameter, public :: PS_DASHDOTDOT = 4
integer(INT_T), parameter, public :: PS_NULL = 5
integer(INT_T), parameter, public :: PS_INSIDEFRAME = 6
integer(INT_T), parameter, public :: PS_USERSTYLE = 7
integer(INT_T), parameter, public :: PS_ALTERNATE = 8

! Background modes (/usr/include/w32api/wingdi.h)
integer(INT_T), parameter, public :: OPAQUE = 1
integer(INT_T), parameter, public :: TRANSPARENT = 1

! Foreground mix modes (/usr/include/w32api/wingdi.h)
integer(INT_T), parameter, public :: R2_BLACK = 1
integer(INT_T), parameter, public :: R2_NOTMERGEPEN = 2
integer(INT_T), parameter, public :: R2_MASKNOTPEN = 3
integer(INT_T), parameter, public :: R2_NOTCOPYPEN = 4
integer(INT_T), parameter, public :: R2_MASKPENNOT = 5
integer(INT_T), parameter, public :: R2_NOT = 6
integer(INT_T), parameter, public :: R2_XORPEN = 7
integer(INT_T), parameter, public :: R2_NOTMASKPEN = 8
integer(INT_T), parameter, public :: R2_MASKPEN = 9
integer(INT_T), parameter, public :: R2_NOTXORPEN = 10
integer(INT_T), parameter, public :: R2_NOP = 11
integer(INT_T), parameter, public :: R2_MERGENOTPEN = 12
integer(INT_T), parameter, public :: R2_COPYPEN = 13
integer(INT_T), parameter, public :: R2_MERGEPENNOT=  14
integer(INT_T), parameter, public :: R2_MERGEPEN = 15
integer(INT_T), parameter, public :: R2_WHITE = 16
integer(INT_T), parameter, public :: R2_LAST = 16

! Raster−operation codes (/usr/include/w32api/wingdi.h)
integer(DWORD_T), parameter, public :: BLACKNESS = 66       ! Z'00000042'
integer(DWORD_T), parameter, public :: SRCCOPY = 13369376   ! Z'00CC0020'
integer(DWORD_T), parameter, public :: WHITENESS = 16711778 ! Z'00FF0062'

! Flags for playing the sound (/usr/include/w32api/mmsystem.h)
integer(DWORD_T), parameter, public :: SND_ALIAS = 65536 ! Z'00010000'

! Show window constants (/usr/include/w32api/winuser.h)
integer(INT_T), parameter, public :: SW_SHOWDEFAULT = 10
integer(INT_T), parameter, public :: SW_SHOW = 5

! Text alignments (/usr/include/w32api/wingdi.h)
integer(INT_T), parameter, public :: TA_NOUPDATECP = 0
integer(INT_T), parameter, public :: TA_UPDATECP = 1
integer(INT_T), parameter, public :: TA_LEFT = 0
integer(INT_T), parameter, public :: TA_RIGHT = 2
integer(INT_T), parameter, public :: TA_CENTER = 6
integer(INT_T), parameter, public :: TA_TOP = 0
integer(INT_T), parameter, public :: TA_BOTTOM = 8
integer(INT_T), parameter, public :: TA_BASELINE = 24
integer(INT_T), parameter, public :: TA_RTLREADING = 256
integer(INT_T), parameter, public :: TA_MASK = &
     (TA_BASELINE+TA_CENTER+TA_UPDATECP+TA_RTLREADING)
integer(INT_T), parameter, public :: VTA_BASELINE = TA_BASELINE
integer(INT_T), parameter, public :: VTA_LEFT = TA_BOTTOM
integer(INT_T), parameter, public :: VTA_RIGHT = TA_TOP
integer(INT_T), parameter, public :: VTA_CENTER = TA_CENTER
integer(INT_T), parameter, public :: VTA_BOTTOM = TA_RIGHT
integer(INT_T), parameter, public :: VTA_TOP = TA_LEFT

! Stock objects brushes (/usr/include/w32api/wingdi.h)
integer(INT_T), parameter, public :: BLACK_BRUSH = 4
integer(INT_T), parameter, public :: DC_BRUSH = 18
integer(INT_T), parameter, public :: DKGRAY_BRUSH = 3
integer(INT_T), parameter, public :: GRAY_BRUSH = 2
integer(INT_T), parameter, public :: HOLLOW_BRUSH = 5
integer(INT_T), parameter, public :: LTGRAY_BRUSH = 1
integer(INT_T), parameter, public :: NULL_BRUSH = 5
integer(INT_T), parameter, public :: OBJ_BRUSH = 2
integer(INT_T), parameter, public :: WHITE_BRUSH = 0
```

```
! Virtual key codes (/usr/include/w32api/winuser.h)
integer(INT_T), parameter, public :: VK_ESCAPE = 27 ! Z'0000001B'

! Windows messages (/usr/include/w32api/winuser.h)
integer(UINT_T), parameter, public :: WM_LBUTTONDOWN = 513 ! Z'00000201'
integer(UINT_T), parameter, public :: WM_CHAR = 258        ! Z'00000102'
integer(UINT_T), parameter, public :: WM_CLOSE = 16        ! Z'00000010'
integer(UINT_T), parameter, public :: WM_COMMAND = 273     ! Z'00000111'
integer(UINT_T), parameter, public :: WM_CREATE = 1        ! Z'00000001'
integer(UINT_T), parameter, public :: WM_DESTROY = 2       ! Z'00000002'
integer(UINT_T), parameter, public :: WM_ERASEBKGND = 20   ! Z'00000014'
integer(UINT_T), parameter, public :: WM_INITDIALOG = 272  ! Z'00000110'
integer(UINT_T), parameter, public :: WM_PAINT = 15        ! Z'0000000F'
integer(UINT_T), parameter, public :: WM_PRINTCLIENT = 792 ! Z'00000318'
integer(UINT_T), parameter, public :: WM_QUIT = 18         ! Z'00000012'
integer(UINT_T), parameter, public :: WM_SIZE = 5          ! Z'00000005'
integer(UINT_T), parameter, public :: WM_TIMER = 275       ! Z'00000113'

! Windows styles (/usr/include/w32api/winuser.h)
integer(DWORD_T), parameter, public :: WS_CAPTION = 12582912   ! Z'00C00000'
integer(DWORD_T), parameter, public :: &
     WS_CLIPCHILDREN = 33554432 ! Z'02000000'
integer(DWORD_T), parameter, public :: &
     WS_CLIPSIBLINGS = 67108864 ! Z'04000000'
integer(DWORD_T), parameter, public :: WS_MAXIMIZEBOX = 65536  ! Z'00010000'
integer(DWORD_T), parameter, public :: WS_MINIMIZEBOX = 131072 ! Z'00020000'
integer(DWORD_T), parameter, public :: WS_SYSMENU = 524288     ! Z'00080000'
integer(DWORD_T), parameter, public :: WS_THICKFRAME = 262144  ! Z'00040000'
integer(DWORD_T), parameter, public :: WS_OVERLAPPED = 0       ! Z'00000000'
integer(DWORD_T), parameter, public :: WS_OVERLAPPEDWINDOW = &
     iany([ WS_OVERLAPPED, WS_CAPTION, WS_SYSMENU, WS_THICKFRAME, &
     WS_MINIMIZEBOX, WS_MAXIMIZEBOX ]) ! 13565952

! Windows styles extended (/usr/include/w32api/winuser.h)
integer(DWORD_T), parameter, public :: WS_EX_CLIENTEDGE = 512  ! Z'00000200'

! ====================
!   WIN32 INTERFACE
! ====================

interface

   function BeginPaint(hWnd,lpPaint) bind(C, name='BeginPaint')
     import :: HDC_T, HWND_T, PAINTSTRUCT_T
     !GCC$ ATTRIBUTES STDCALL :: BeginPaint
     integer(HDC_T) :: BeginPaint
     integer(HWND_T), value :: hWnd
     type(PAINTSTRUCT_T), intent(out) :: lpPaint
   end function BeginPaint

   function BitBlt(hdcDest,nXDest,nYDest,nWidth,nHeight,hdcSrc, &
        nXSrc,nYSrc,dwRop) bind(C, name='BitBlt')
     import :: BOOL_T, DWORD_T, HDC_T, INT_T
     !GCC$ ATTRIBUTES STDCALL :: BitBlt
     integer(BOOL_T) :: BitBlt
     integer(HDC_T), value :: hdcDest
     integer(INT_T), value :: nXDest
     integer(INT_T), value :: nYDest
     integer(INT_T), value :: nWidth
     integer(INT_T), value :: nHeight
     integer(HDC_T), value :: hdcSrc
     integer(INT_T), value :: nXSrc
     integer(INT_T), value :: nYSrc
     integer(DWORD_T), value :: dwRop
   end function BitBlt

   function CheckRadioButton(hDlg,nIDFirstButton,nIDLastButton, &
        nIDCheckButton) bind(C, name='CheckRadioButton')
     import :: BOOL_T, HWND_T, INT_T
     !GCC$ ATTRIBUTES STDCALL :: CheckRadioButton
     integer(BOOL_T) :: CheckRadioButton
     integer(HWND_T), value :: hDlg
     integer(INT_T), value :: nIDFirstButton
     integer(INT_T), value :: nIDLastButton
     integer(INT_T), value :: nIDCheckButton
   end function CheckRadioButton
```

```
function CreateCompatibleBitmap(hdc,nWidth,nHeight) &
     bind(C, name='CreateCompatibleBitmap')
  import :: HBITMAP_T, HDC_T, INT_T
  !GCC$ ATTRIBUTES STDCALL :: CreateCompatibleBitmap
  integer(HBITMAP_T) :: CreateCompatibleBitmap
  integer(HDC_T), value :: hdc
  integer(INT_T), value :: nWidth
  integer(INT_T), value :: nHeight
end function CreateCompatibleBitmap

function CreateCompatibleDC(hdc) bind(C, name='CreateCompatibleDC')
  import :: HDC_T
  !GCC$ ATTRIBUTES STDCALL :: CreateCompatibleDC
  integer(HDC_T) :: CreateCompatibleDC
  integer(HDC_T), value :: hdc
end function CreateCompatibleDC

function CreatePen(fnPenStyle,nWidth,crColor) bind(C, name='CreatePen')
  import :: COLORREF_T, HPEN_T, INT_T
  !GCC$ ATTRIBUTES STDCALL :: CreatePen
  integer(HPEN_T) :: CreatePen
  integer(INT_T), value :: fnPenStyle
  integer(INT_T), value :: nWidth
  integer(COLORREF_T), value :: crColor
end function CreatePen

function CreateSolidBrush(crColor) bind(C, name='CreateSolidBrush')
  import :: COLORREF_T, HBRUSH_T
  !GCC$ ATTRIBUTES STDCALL :: CreateSolidBrush
  integer(HBRUSH_T) :: CreateSolidBrush
  integer(COLORREF_T), value :: crColor
end function CreateSolidBrush

function CreateHatchBrush(fnStyle,clrref) bind(C, name='CreateHatchBrush')
  import :: COLORREF_T, HBRUSH_T, INT_T
  !GCC$ ATTRIBUTES STDCALL :: CreateHatchBrush
  integer(HBRUSH_T) :: CreateHatchBrush
  integer(INT_T), value :: fnStyle
  integer(COLORREF_T), value :: clrref
end function CreateHatchBrush

function CreateWindowEx(dwExStyle,lpClassName,lpWindowName,dwStyle, &
     x,y,nWidth,nHeight, &
     hWndParent,hMenu,hInstance,lpParam) bind(C, name='CreateWindowExA')
  import :: C_CHAR, C_PTR, DWORD_T, HINSTANCE_T, HMENU_T, HWND_T, INT_T
  !GCC$ ATTRIBUTES STDCALL :: CreateWindowEx
  integer(HWND_T) :: CreateWindowEx
  integer(DWORD_T), value :: dwExStyle
  character(C_CHAR), intent(in) :: lpClassName(*)  ! LPCTSTR
  character(C_CHAR), intent(in) :: lpWindowName(*) ! LPCTSTR
  integer(DWORD_T), value :: dwStyle
  integer(INT_T), value :: x
  integer(INT_T), value :: y
  integer(INT_T), value :: nWidth
  integer(INT_T), value :: nHeight
  integer(HWND_T), value :: hWndParent
  integer(HMENU_T), value :: hMenu
  integer(HINSTANCE_T), value :: hInstance
  type(C_PTR), value :: lpParam
end function CreateWindowEx

function DeleteDC(hdc) bind(C, name='DeleteDC')
  import :: BOOL_T, HDC_T
  !GCC$ ATTRIBUTES STDCALL :: DeleteDC
  integer(BOOL_T) :: DeleteDC
  integer(HDC_T), value :: hdc
end function DeleteDC

function DeleteObject(hObject) bind(C, name='DeleteObject')
  import :: BOOL_T, HGDIOBJ_T
  !GCC$ ATTRIBUTES STDCALL :: DeleteObject
  integer(BOOL_T) :: DeleteObject
  integer(HGDIOBJ_T), value :: hObject
end function DeleteObject

function DestroyWindow(hWnd) bind(C, name='DestroyWindow')
  import :: BOOL_T, HWND_T
  !GCC$ ATTRIBUTES STDCALL :: DestroyWindow
```

```
  integer(BOOL_T) :: DestroyWindow
  integer(HWND_T), value :: hWnd
end function DestroyWindow

function DefWindowProc(hWnd,Msg,wParam,lParam) &
     bind(C, name='DefWindowProcA')
  import :: HWND_T, LPARAM_T, LRESULT_T, UINT_T, WPARAM_T
  !GCC$ ATTRIBUTES STDCALL :: DefWindowProc
  integer(LRESULT_T) :: DefWindowProc
  integer(HWND_T), value :: hWnd
  integer(UINT_T), value :: Msg
  integer(WPARAM_T), value :: wParam
  integer(LPARAM_T), value :: lParam
end function DefWindowProc

function DialogBoxParam(hInstance,lpTemplate,hWndParent,lpDialogFunc, &
     dwInitParam) bind(C, name='DialogBoxParamA')
  import :: C_CHAR, C_FUNPTR, HINSTANCE_T, HWND_T, INT_PTR_T, LPARAM_T
  !GCC$ ATTRIBUTES STDCALL :: DialogBoxParam
  integer(INT_PTR_T) :: DialogBoxParam
  integer(HINSTANCE_T), value :: hInstance
  character(C_CHAR), intent(in) :: lpTemplate(*) ! LPCTSTR
  !type(C_PTR), value :: lpTemplate ! LPCTSTR
  integer(HWND_T), value :: hWndParent
  type(C_FUNPTR), value :: lpDialogFunc ! DLGPROC
  integer(LPARAM_T), value :: dwInitParam
end function DialogBoxParam

function DispatchMessage(lpMsg) bind(C, name='DispatchMessageA')
  import :: LRESULT_T, MSG_T
  !GCC$ ATTRIBUTES STDCALL :: DispatchMessage
  integer(LRESULT_T) :: DispatchMessage
  type(MSG_T), intent(in) :: lpMsg
end function DispatchMessage

function DrawText(hdc,lpString,nCount,lpRect,uFormat) &
     bind(C, name='DrawTextA')
  import :: C_CHAR, HDC_T, INT_T, RECT_T, UINT_T
  !GCC$ ATTRIBUTES STDCALL :: DrawText
  integer(INT_T) :: DrawText
  integer(HDC_T), value :: hdc
  character(C_CHAR), intent(inout) :: lpString(*) ! LPCTSTR
  integer(INT_T), value :: nCount
  type(RECT_T), intent(inout) :: lpRect
  integer(UINT_T), value :: uFormat
end function DrawText

function Ellipse(hdc,nLeftRect,nTopRect,nRightRect,nBottomRect) &
     bind(C, name='Ellipse')
  import :: BOOL_T, HDC_T, INT_T
  !GCC$ ATTRIBUTES STDCALL :: Ellipse
  integer(BOOL_T) :: Ellipse
  integer(HDC_T), value :: hdc
  integer(INT_T), value :: nLeftRect
  integer(INT_T), value :: nTopRect
  integer(INT_T), value :: nRightRect
  integer(INT_T), value :: nBottomRect
end function Ellipse

function EndDialog(hWnd,nResult) bind(C, name='EndDialog')
  import :: BOOL_T, HWND_T, INT_PTR_T
  !GCC$ ATTRIBUTES STDCALL :: EndDialog
  integer(BOOL_T) :: EndDialog
  integer(HWND_T), value :: hWnd
  integer(INT_PTR_T), value :: nResult
end function EndDialog

function EndPaint(hWnd,lpPaint) bind(C, name='EndPaint')
  import :: BOOL_T, HWND_T, PAINTSTRUCT_T
  !GCC$ ATTRIBUTES STDCALL :: EndPaint
  integer(BOOL_T) :: EndPaint
  integer(HWND_T), value :: hWnd
  type(PAINTSTRUCT_T), intent(in) :: lpPaint
end function EndPaint

subroutine ExitProcess(uExitCode) bind(C, name='ExitProcess')
  import :: UINT_T
  !GCC$ ATTRIBUTES STDCALL :: ExitProcess
```

```
  integer(UINT_T), value :: uExitCode
end subroutine ExitProcess

function FillRect(hdc,lprc,hbr) bind(C, name='FillRect')
  import :: HBRUSH_T, HDC_T, INT_T, RECT_T
  !GCC$ ATTRIBUTES STDCALL :: FillRect
  integer(INT_T) :: FillRect
  integer(HDC_T),value :: hdc
  type(RECT_T), intent(in) :: lprc
  integer(HBRUSH_T), value :: hbr
end function FillRect

function GetBkColor(hdc) bind(C, name='GetBkColor')
  import :: COLORREF_T, HDC_T
  !GCC$ ATTRIBUTES STDCALL :: GetBkColor
  integer(COLORREF_T) :: GetBkColor
  integer(HDC_T), value :: hdc
end function GetBkColor

function GetClientRect(hWnd,lpRect) bind(C, name='GetClientRect')
  import :: BOOL_T, HWND_T, RECT_T
  !GCC$ ATTRIBUTES STDCALL :: GetClientRect
  integer(BOOL_T) :: GetClientRect
  integer(HWND_T), value :: hWnd
  type(RECT_T), intent(out) :: lpRect
end function GetClientRect

function GetCommandLine() bind(C, name='GetCommandLineA')
  import :: C_PTR
  !GCC$ ATTRIBUTES STDCALL :: GetCommandLine
  type(C_PTR) :: GetCommandLine ! LPCTSTR
end function GetCommandLine

function GetDC(hWnd) bind(C, name='GetDC')
  import :: HDC_T, HWND_T
  !GCC$ ATTRIBUTES STDCALL :: GetDC
  integer(HDC_T) :: GetDC
  integer(HWND_T), value :: hWnd
end function GetDC

function GetDCBrushColor(hdc) bind(C, name='GetDCBrushColor')
  import :: COLORREF_T, HDC_T
  !GCC$ ATTRIBUTES STDCALL :: GetDCBrushColor
  integer(COLORREF_T) :: GetDCBrushColor
  integer(HDC_T), value :: hdc
end function GetDCBrushColor

function GetDeviceCaps(hdc,nIndex) bind(C, name='GetDeviceCaps')
  import :: HDC_T, INT_T
  !GCC$ ATTRIBUTES STDCALL :: GetDeviceCaps
  integer(INT_T) :: GetDeviceCaps
  integer(HDC_T), value :: hdc
  integer(INT_T), value :: nIndex
end function GetDeviceCaps

function GetDlgItemText(hDlg,nIDDlgItem,lpString,nMaxCount) &
     bind(C, name='GetDlgItemTextA')
  import :: C_CHAR, HWND_T, INT_T, UINT_T
  !GCC$ ATTRIBUTES STDCALL :: GetDlgItemText
  integer(UINT_T) :: GetDlgItemText
  integer(HWND_T), value :: hDlg
  integer(INT_T), value :: nIDDlgItem
  character(C_CHAR), intent(out) :: lpString(*) ! LPTSTR
  ! This works too... but it is more complicated :-(
  ! Notice that the C string pointer is of type value:
  ! it is the content to which it points that is an 'output'
  !
  !type(C_PTR), value :: lpString
  integer(INT_T), value :: nMaxCount
end function GetDlgItemText

function GetKeyState(nVirtKey) bind(C, name='GetKeyState')
  import :: INT_T, SHORT_T
  !GCC$ ATTRIBUTES STDCALL :: GetKeyState
  integer(SHORT_T) :: GetKeyState
  integer(INT_T), value :: nVirtKey
end function GetKeyState
```

```
function GetLastError() bind(C, name='GetLastError')
  import :: DWORD_T
  !GCC$ ATTRIBUTES STDCALL :: GetLastError
  integer(DWORD_T) :: GetLastError
end function GetLastError

function GetMessage(lpMsg,hWnd,wMsgFilterMin,wMsgFilterMax) &
     bind(C, name='GetMessageA')
  import :: BOOL_T, HWND_T, MSG_T, UINT_T
  !GCC$ ATTRIBUTES STDCALL :: GetMessage
  integer(BOOL_T) :: GetMessage
  type(MSG_T), intent(out) :: lpMsg
  integer(HWND_T), value :: hWnd
  integer(UINT_T), value :: wMsgFilterMin
  integer(UINT_T), value :: wMsgFilterMax
end function GetMessage

function GetModuleHandle(lpModuleName) bind(C, name='GetModuleHandleA')
  import :: C_CHAR, HMODULE_T
  !GCC$ ATTRIBUTES STDCALL :: GetModuleHandle
  integer(HMODULE_T) :: GetModuleHandle
  character(C_CHAR), intent(in) :: lpModuleName(*) ! LPCTSTR
end function GetModuleHandle

function GetMonitorInfo(hMonitor,lpmi) bind(C, name='GetMonitorInfoA')
  import :: BOOL_T, HMONITOR_T, MONITORINFO_T
  !GCC$ ATTRIBUTES STDCALL :: GetMonitorInfo
  integer(BOOL_T) :: GetMonitorInfo
  integer(HMONITOR_T), value :: hMonitor
  type(MONITORINFO_T), intent(out) :: lpmi
end function GetMonitorInfo

function GetStockObject(fnObject) bind(C, name='GetStockObject')
  import :: HGDIOBJ_T, INT_T
  !GCC$ ATTRIBUTES STDCALL :: GetStockObject
  integer(HGDIOBJ_T) :: GetStockObject
  integer(INT_T), value :: fnObject
end function GetStockObject

function GetTextColor(hdc) bind(C, name='GetTextColor')
  import :: COLORREF_T, HDC_T
  !GCC$ ATTRIBUTES STDCALL :: GetTextColor
  integer(COLORREF_T) :: GetTextColor
  integer(HDC_T), value :: hdc
end function GetTextColor

function InvalidateRect(hWnd,lpRect,bErase) bind(C, name='InvalidateRect')
  import :: BOOL_T, HWND_T, RECT_T
  !GCC$ ATTRIBUTES STDCALL :: InvalidateRect
  integer(BOOL_T) :: InvalidateRect
  integer(HWND_T), value :: hWnd
  type(RECT_T), intent(in) :: lpRect
  integer(BOOL_T), value :: bErase
end function InvalidateRect

function KillTimer(hWnd,uIDEvent) bind(C, name='KillTimer')
  import :: BOOL_T, HWND_T, UINT_PTR_T
  !GCC$ ATTRIBUTES STDCALL :: KillTimer
  integer(BOOL_T) :: KillTimer
  integer(HWND_T), value :: hWnd
  integer(UINT_PTR_T), value :: uIDEvent
end function KillTimer

function LineTo(hdc,nXEnd,nYEnd) bind(C, name='LineTo')
  import :: BOOL_T, HDC_T, INT_T
  !GCC$ ATTRIBUTES STDCALL :: LineTo
  integer(BOOL_T) :: LineTo
  integer(HDC_T),value :: hdc
  integer(INT_T), value :: nXEnd
  integer(INT_T), value :: nYEnd
end function LineTo

function LoadCursor(hInstance,lpCursorName) bind(C, name='LoadCursorA')
  import :: C_CHAR, HCURSOR_T, HINSTANCE_T
  !GCC$ ATTRIBUTES STDCALL :: LoadCursor
  integer(HCURSOR_T) :: LoadCursor
  integer(HINSTANCE_T), value :: hInstance
  character(C_CHAR), intent(in) :: lpCursorName(*) ! LPCTSTR
```

```
end function LoadCursor

function LoadIcon(hInstance,lpIconName) bind(C, name='LoadIconA')
  import :: C_CHAR, HICON_T, HINSTANCE_T
  !GCC$ ATTRIBUTES STDCALL :: LoadIcon
  integer(HICON_T) :: LoadIcon
  integer(HINSTANCE_T), value :: hInstance
  character(C_CHAR), intent(in) :: lpIconName(*) ! LPCTSTR
end function LoadIcon

function MessageBeep(uType) bind(C, name='MessageBeep')
  import :: BOOL_T, UINT_T
  !GCC$ ATTRIBUTES STDCALL :: MessageBeep
  integer(BOOL_T) :: MessageBeep
  integer(UINT_T), value :: uType
end function MessageBeep

function MessageBox(hWnd,lpText,lpCaption,uType) &
     bind(C, name='MessageBoxA')
  import :: C_CHAR, HWND_T, INT_T, UINT_T
  !GCC$ ATTRIBUTES STDCALL :: MessageBox
  integer(INT_T) :: MessageBox
  integer(HWND_T), value :: hWnd
  character(C_CHAR), intent(in) :: lpText(*)    ! LPCTSTR
  character(C_CHAR), intent(in) :: lpCaption(*) ! LPCTSTR
  integer(UINT_T), value :: uType
end function MessageBox

function MonitorFromPoint(pt,dwFlags) bind(C, name='MonitorFromPoint')
  import :: DWORD_T, HMONITOR_T, POINT_T
  !GCC$ ATTRIBUTES STDCALL :: MonitorFromPoint
  integer(HMONITOR_T) :: MonitorFromPoint
  type(POINT_T), value :: pt
  integer(DWORD_T), value :: dwFlags
end function MonitorFromPoint

function MoveToEx(hdc,X,Y,lpPoint) bind(C, name='MoveToEx')
  import :: BOOL_T, HDC_T, INT_T, POINT_T
  !GCC$ ATTRIBUTES STDCALL :: MoveToEx
  integer(BOOL_T) :: MoveToEx
  integer(HDC_T),value :: hdc
  integer(INT_T), value :: X
  integer(INT_T), value :: Y
  type(POINT_T), intent(out) :: lpPoint
end function MoveToEx

function PlaySound(pszSound,hmod,fdwSound) bind(C, name='PlaySoundA')
  import :: BOOL_T, C_CHAR, DWORD_T, HMODULE_T
  !GCC$ ATTRIBUTES STDCALL :: PlaySound
  integer(BOOL_T) :: PlaySound
  character(C_CHAR), intent(in) :: pszSound(*) ! LPCTSTR
  integer(HMODULE_T), value :: hmod
  integer(DWORD_T), value :: fdwSound
end function PlaySound

function PeekMessage(lpMsg,hWnd,wMsgFilterMin,wMsgFilterMax,wRemoveMsg) &
     bind(C, name='PeekMessageA')
  import :: BOOL_T, HWND_T, MSG_T, UINT_T
  !GCC$ ATTRIBUTES STDCALL :: PeekMessage
  integer(BOOL_T) :: PeekMessage
  type(MSG_T), intent(out) :: lpMsg
  integer(HWND_T), value :: hWnd
  integer(UINT_T), value :: wMsgFilterMin
  integer(UINT_T), value :: wMsgFilterMax
  integer(UINT_T), value :: wRemoveMsg
end function PeekMessage

function Polygon(hdc,lpPoints,nCount) bind(C, name='Polygon')
  import :: BOOL_T, HDC_T, INT_T, POINT_T
  !GCC$ ATTRIBUTES STDCALL :: Polygon
  integer(BOOL_T) :: Polygon
  integer(HDC_T),value :: hdc
  type(POINT_T), intent(in) :: lpPoints(*)
  integer(INT_T), value :: nCount
end function Polygon

function PostMessage(hWnd,Msg,wParam,lParam) bind(C, name='PostMessageA')
  import :: BOOL_T, HWND_T, LPARAM_T, UINT_T, WPARAM_T
```

```
  !GCC$ ATTRIBUTES STDCALL :: PostMessage
  integer(BOOL_T) :: PostMessage
  integer(HWND_T), value :: hWnd
  integer(UINT_T), value :: Msg
  integer(WPARAM_T), value :: wParam
  integer(LPARAM_T), value :: lParam
end function PostMessage

subroutine PostQuitMessage(nExitCode) bind(C, name='PostQuitMessage')
  import :: INT_T
  !GCC$ ATTRIBUTES STDCALL :: PostQuitMessage
  integer(INT_T), value :: nExitCode
end subroutine PostQuitMessage

function Rectangle(hdc,nLeftRect,nTopRect,nRightRect,nBottomRect) &
     bind(C, name='Rectangle')
  import :: BOOL_T, HDC_T, INT_T
  !GCC$ ATTRIBUTES STDCALL :: Rectangle
  integer(BOOL_T) :: Rectangle
  integer(HDC_T),value :: hdc
  integer(INT_T), value :: nLeftRect
  integer(INT_T), value :: nTopRect
  integer(INT_T), value :: nRightRect
  integer(INT_T), value :: nBottomRect
end function Rectangle

function RegisterClassEx(WndClass) bind(C, name='RegisterClassExA')
  import :: ATOM_T, WNDCLASSEX_T
  !GCC$ ATTRIBUTES STDCALL :: RegisterClassEx
  integer(ATOM_T) :: RegisterClassEx
  type(WNDCLASSEX_T), intent(in) :: WndClass
end function RegisterClassEx

function ReleaseDC(hWnd,hdc) bind(C, name='ReleaseDC')
  import :: HDC_T, HWND_T, INT_T
  !GCC$ ATTRIBUTES STDCALL :: ReleaseDC
  integer(INT_T) :: ReleaseDC
  integer(HWND_T), value :: hWnd
  integer(HDC_T), value :: hdc
end function ReleaseDC

function SelectObject(hdc,hgdiobj) bind(C, name='SelectObject')
  import :: HDC_T, HGDIOBJ_T
  !GCC$ ATTRIBUTES STDCALL :: SelectObject
  integer(HGDIOBJ_T) :: SelectObject
  integer(HDC_T), value :: hdc
  integer(HGDIOBJ_T), value :: hgdiobj
end function SelectObject

function SendMessage(hWnd,Msg,wParam,lParam) bind(C, name='SendMessageA')
  import :: HWND_T, LPARAM_T, LRESULT_T, UINT_T, WPARAM_T
  !GCC$ ATTRIBUTES STDCALL :: SendMessage
  integer(LRESULT_T) :: SendMessage
  integer(HWND_T), value :: hWnd
  integer(UINT_T), value :: Msg
  integer(WPARAM_T), value :: wParam
  integer(LPARAM_T), value :: lParam
end function SendMessage

function SendNotifyMessage(hWnd,Msg,wParam,lParam) &
     bind(C, name='SendNotifyMessageA')
  import :: HWND_T, LPARAM_T, LRESULT_T, UINT_T, WPARAM_T
  !GCC$ ATTRIBUTES STDCALL :: SendNotifyMessage
  integer(LRESULT_T) :: SendNotifyMessage
  integer(HWND_T), value :: hWnd
  integer(UINT_T), value :: Msg
  integer(WPARAM_T), value :: wParam
  integer(LPARAM_T), value :: lParam
end function SendNotifyMessage

function SetBkColor(hdc,crColor) bind(C, name='SetBkColor')
  import :: COLORREF_T, HDC_T
  !GCC$ ATTRIBUTES STDCALL :: SetBkColor
  integer(COLORREF_T) :: SetBkColor
  integer(HDC_T), value :: hdc
  integer(COLORREF_T), value :: crColor
end function SetBkColor
```

```
function SetBkMode(hdc,iBkMode) bind(C, name='SetBkMode')
  import :: HDC_T, INT_T
  !GCC$ ATTRIBUTES STDCALL :: SetBkMode
  integer(INT_T) :: SetBkMode
  integer(HDC_T), value :: hdc
  integer(INT_T), value :: iBkMode
end function SetBkMode

function SetCursor(hCursor) bind(C, name='SetCursor')
  import :: HCURSOR_T
  !GCC$ ATTRIBUTES STDCALL :: SetCursor
  integer(HCURSOR_T) :: SetCursor
  integer(HCURSOR_T), value :: hCursor
end function SetCursor

function SetDCBrushColor(hdc,crColor) bind(C, name='SetDCBrushColor')
  import :: COLORREF_T, HDC_T
  !GCC$ ATTRIBUTES STDCALL :: SetDCBrushColor
  integer(COLORREF_T) :: SetDCBrushColor
  integer(HDC_T), value :: hdc
  integer(COLORREF_T), value :: crColor
end function SetDCBrushColor

function SetDlgItemText(hDlg,nIDDlgItem,lpString) &
     bind(C, name='SetDlgItemTextA')
  import :: BOOL_T, C_CHAR, HWND_T, INT_T
  !GCC$ ATTRIBUTES STDCALL :: SetDlgItemText
  integer(BOOL_T) :: SetDlgItemText
  integer(HWND_T), value :: hDlg
  integer(INT_T), value :: nIDDlgItem
  character(C_CHAR), intent(in) :: lpString(*) ! LPCTSTR
end function SetDlgItemText

function SetPixel(hdc,X,Y,crColor) bind(C, name='SetPixel')
  import :: COLORREF_T, HDC_T, INT_T
  !GCC$ ATTRIBUTES STDCALL :: SetPixel
  integer(COLORREF_T) :: SetPixel
  integer(HDC_T), value :: hdc
  integer(INT_T), value :: X
  integer(INT_T), value :: Y
  integer(COLORREF_T), value :: crColor
end function SetPixel

function SetROP2(hdc,fnDrawMode) bind(C, name='SetROP2')
  import :: HDC_T, INT_T
  !GCC$ ATTRIBUTES STDCALL :: SetROP2
  integer(INT_T) :: SetROP2
  integer(HDC_T), value :: hdc
  integer(INT_T), value :: fnDrawMode
end function SetROP2

function SetTextAlign(hdc,fMode) bind(C, name='SetTextAlign')
  import :: HDC_T, UINT_T
  !GCC$ ATTRIBUTES STDCALL :: SetTextAlign
  integer(UINT_T) :: SetTextAlign
  integer(HDC_T), value :: hdc
  integer(UINT_T), value :: fMode
end function SetTextAlign

function SetTextColor(hdc,crColor) bind(C, name='SetTextColor')
  import :: COLORREF_T, HDC_T
  !GCC$ ATTRIBUTES STDCALL :: SetTextColor
  integer(COLORREF_T) :: SetTextColor
  integer(HDC_T), value :: hdc
  integer(COLORREF_T), value :: crColor
end function SetTextColor

function SetTimer(hWnd,nIDEvent,uElapse,lpTimerFunc) &
     bind(C,name='SetTimer')
  import :: C_FUNPTR, HWND_T, UINT_T, UINT_PTR_T
  !GCC$ ATTRIBUTES STDCALL :: SetTimer
  integer(UINT_PTR_T) :: SetTimer
  integer(HWND_T), value :: hWnd
  integer(UINT_PTR_T), value :: nIDEvent
  integer(UINT_T), value :: uElapse
  type(C_FUNPTR), value :: lpTimerFunc ! TIMERPROC
end function SetTimer
```

```
    function ShowWindow(hWnd,nCmdShow) bind(C, name='ShowWindow')
      import :: BOOL_T, HWND_T, INT_T
      !GCC$ ATTRIBUTES STDCALL :: ShowWindow
      integer(BOOL_T) :: ShowWindow
      integer(HWND_T), value :: hWnd
      integer(INT_T), value :: nCmdShow
    end function ShowWindow

    function TextOut(hdc,nXStart,nYStart,lpString,cchString) &
        bind(C, name='TextOutA')
      import :: BOOL_T, C_CHAR, HDC_T, INT_T
      !GCC$ ATTRIBUTES STDCALL :: TextOut
      integer(BOOL_T) :: TextOut
      integer(HDC_T), value :: hdc
      integer(INT_T), value :: nXStart
      integer(INT_T), value :: nYStart
      character(C_CHAR), intent(in) :: lpString(*) ! LPCTSTR
      integer(INT_T), value :: cchString
    end function TextOut

    function TranslateMessage(lpMsg) bind(C, name='TranslateMessage')
      import :: BOOL_T, MSG_T
      !GCC$ ATTRIBUTES STDCALL :: TranslateMessage
      integer(BOOL_T) :: TranslateMessage
      type(MSG_T), intent(in) :: lpMsg
    end function TranslateMessage

    function UpdateWindow(hWnd) bind(C, name='UpdateWindow')
      import :: BOOL_T, HWND_T
      !GCC$ ATTRIBUTES STDCALL :: UpdateWindow
      integer(BOOL_T) :: UpdateWindow
      integer(HWND_T), value :: hWnd
    end function UpdateWindow
  end interface

  ! Interface routines
  public :: BeginPaint, BitBlt, CheckRadioButton, CreateCompatibleBitmap, &
       CreateCompatibleDC, CreatePen, CreateSolidBrush, CreateHatchBrush, &
       CreateWindowEx, DeleteDC, DeleteObject, DestroyWindow, DefWindowProc, &
       DialogBoxParam, DispatchMessage, DrawText, Ellipse, EndDialog, &
       EndPaint, ExitProcess, FillRect, GetBkColor, GetClientRect, &
       GetCommandLine, GetDC, GetDCBrushColor, GetDeviceCaps, GetDlgItemText, &
       GetKeyState, GetLastError, GetMessage, GetModuleHandle, &
       GetMonitorInfo, GetStockObject, GetTextColor, InvalidateRect, &
       KillTimer, LineTo, LoadCursor, LoadIcon, MessageBeep, MessageBox, &
       MonitorFromPoint, MoveToEx, PlaySound, PeekMessage, Polygon, &
       PostMessage, PostQuitMessage, Rectangle, RegisterClassEx, ReleaseDC, &
       SelectObject, SendMessage, SendNotifyMessage, SetBkColor, SetBkMode, &
       SetCursor, SetDCBrushColor, SetDlgItemText, SetPixel, SetROP2, &
       SetTextAlign, SetTextColor, SetTimer, ShowWindow, TextOut, &
       TranslateMessage, UpdateWindow

  ! Auxiliary routines
  public :: arrow_cursor, cross_cursor, hand_cursor, wait_cursor, &
       application_icon, asterisk_icon, error_icon, exclamation_icon, &
       hand_icon, information_icon, question_icon, warning_icon, winlogo_icon, &
       ask_confirmation, dialog_box, hi_word, lo_word, make_int_resource, &
       make_int_resource_C_PTR, null_p, RGB, error_msg

contains

  function arrow_cursor() result(s)
    character(C_CHAR), pointer :: s(:)
    s => make_int_resource(IDC_ARROW)
  end function arrow_cursor

  function cross_cursor() result(s)
    character(C_CHAR), pointer :: s(:)
    s => make_int_resource(IDC_CROSS)
  end function cross_cursor

  function hand_cursor() result(s)
    character(C_CHAR), pointer :: s(:)
    s => make_int_resource(IDC_HAND)
  end function hand_cursor

  function wait_cursor() result(s)
    character(C_CHAR), pointer :: s(:)
```

```
  s => make_int_resource(IDC_WAIT)
end function wait_cursor

function application_icon() result(s)
  character(C_CHAR), pointer :: s(:)
  s => make_int_resource(IDI_APPLICATION)
end function application_icon

function asterisk_icon() result(s)
  character(C_CHAR), pointer :: s(:)
  s => make_int_resource(IDI_ASTERISK)
end function asterisk_icon

function error_icon() result(s)
  character(C_CHAR), pointer :: s(:)
  s => make_int_resource(IDI_ERROR)
end function error_icon

function exclamation_icon() result(s)
  character(C_CHAR), pointer :: s(:)
  s => make_int_resource(IDI_EXCLAMATION)
end function exclamation_icon

function hand_icon() result(s)
  character(C_CHAR), pointer :: s(:)
  s => make_int_resource(IDI_HAND)
end function hand_icon

function information_icon() result(s)
  character(C_CHAR), pointer :: s(:)
  s => make_int_resource(IDI_INFORMATION)
end function information_icon

function question_icon() result(s)
  character(C_CHAR), pointer :: s(:)
  s => make_int_resource(IDI_QUESTION)
end function question_icon

function warning_icon() result(s)
  character(C_CHAR), pointer :: s(:)
  s => make_int_resource(IDI_WARNING)
end function warning_icon

function winlogo_icon() result(s)
  character(C_CHAR), pointer :: s(:)
  s => make_int_resource(IDI_WINLOGO)
end function winlogo_icon

function ask_confirmation(hWnd,lpText,lpCaption)
  integer(INT_T) :: ask_confirmation
  integer(HWND_T), intent(in) :: hWnd
  character(C_CHAR), intent(in) :: lpText(*) ! LPCTSTR
  character(C_CHAR), intent(in) :: lpCaption(*) ! LPCTSTR
  ask_confirmation = MessageBox(hWnd,lpText,lpCaption, &
       ior(MB_YESNO,MB_ICONQUESTION))
end function ask_confirmation

function dialog_box(hInstance,lpTemplate,hWndParent,lpDialogFunc)
  integer(INT_PTR_T) :: dialog_box
  integer(HINSTANCE_T), intent(in) :: hInstance
  character(C_CHAR), intent(in) :: lpTemplate(*) ! LPCTSTR
  !type(C_PTR), intent(in) :: lpTemplate ! LPCTSTR
  integer(HWND_T), intent(in) :: hWndParent
  type(C_FUNPTR), intent(in) :: lpDialogFunc ! DLGPROC
  dialog_box = DialogBoxParam(hInstance,lpTemplate,hWndParent, &
       lpDialogFunc,NULL_T)
end function dialog_box

function hi_word(dwValue)
  integer(WORD_T) :: hi_word
  integer(DWORD_T), intent(in) :: dwValue
  ! (/usr/include/w32api/windef.h)
  hi_word = int(ishft(dwValue,-16),WORD_T)
end function hi_word

function lo_word(dwValue)
  integer(WORD_T) :: lo_word
  integer(DWORD_T), intent(in) :: dwValue
```

```
    ! (/usr/include/w32api/windef.h)
    lo_word = int(iand(dwValue,65535),WORD_T) ! iand(dwValue,Z'0000FFFF')
  end function lo_word

  function make_int_resource(i) result(s)
    integer(WORD_T), intent(in) :: i
    character(C_CHAR), pointer :: s(:) ! LPTSTR
    call c_f_pointer(make_int_resource_C_PTR(i),s,[0])
  end function make_int_resource

  function make_int_resource_C_PTR(i) result(s)
    integer(WORD_T), intent(in) :: i
    type(C_PTR) :: s
    s = transfer(int(i,HANDLE_T),NULL_PTR_T)
  end function make_int_resource_C_PTR

  function null_p() result(s)
    character(C_CHAR), pointer :: s(:) ! LPTSTR
    s => make_int_resource(0_WORD_T)
  end function null_p

  function RGB(r,g,b)
    integer(COLORREF_T) :: RGB
    integer(INT_T), intent(in) :: r, g, b
    RGB = (ior(ior((r),ishft((g),8)),ishft((b),16)))
  end function RGB

  subroutine error_msg(lpText)
    character(C_CHAR), intent(in) :: lpText(*) ! LPCTSTR
    integer :: dummy
    dummy = MessageBox(NULL_T,lpText,NULL_LPSTR,ior(MB_ICONEXCLAMATION,MB_OK))
  end subroutine error_msg
end module win32
```

```
!
! (Partial) Fortran Interface to the Windows API Library
! by Angelo Graziosi  (firstname.lastnameATalice.it)
! Copyright Angelo Graziosi
!
! It is distributed in the hope that it will be useful,
! but WITHOUT ANY WARRANTY; without even the implied warranty of
! MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
!
! DESCRIPTION
!   win32 boxes (aka, dialogues) modules...
!   Just to start with Windows applications...
!
!   From: my C++ Windows Applications and Borland C++ 2.0 examples
!
!   Notice that:
!
!       int(0,UINT_T)    -->  0_UINT_T
!       int(0,WPARAM_T)  -->  0_WPARAM_T
!       int(0,LPARAM_T)  -->  0_LPARAM_T
!       ...
!

module AboutBox_class
  use, intrinsic :: iso_c_binding, only: c_funloc
  use win32, only: BOOL_T, DWORD_T, FALSE_T, HWND_T, IDCANCEL, IDOK, &
       INT_PTR_T, LPARAM_T, NULL_LPSTR, TRUE_T, UINT_T, WM_COMMAND, &
       WM_INITDIALOG, WORD_T, WPARAM_T, &
       dialog_box, EndDialog, GetModuleHandle, lo_word, make_int_resource
  implicit none
  private

  type, public :: AboutBox
     private
     integer(HWND_T) :: hDlg
     integer(WORD_T) :: idd_about
  end type AboutBox

  ! TRUE_T if OK button is pressed, otherwise it is FALSE_T
  integer(BOOL_T) :: dialog_result = FALSE_T

  interface new_box
     module procedure AboutBox_init
  end interface new_box

  interface run
     module procedure AboutBox_run
  end interface run

  public :: new_box, run

contains

  subroutine AboutBox_init(this,hhDlg,idd_ab)
    type(AboutBox), intent(out) :: this
    integer(HWND_T), intent(in) :: hhDlg
    integer(WORD_T), intent(in) :: idd_ab

    this%hDlg = hhDlg
    this%idd_about = idd_ab
  end subroutine AboutBox_init

  function AboutBox_run(this)
    integer(BOOL_T) :: AboutBox_run
    type(AboutBox), intent(in) :: this
    integer(INT_PTR_T) :: dummy

    dummy = dialog_box(GetModuleHandle(NULL_LPSTR), &
         make_int_resource(this%idd_about),this%hDlg,c_funloc(AboutDlgProc))

    AboutBox_run = dialog_result
  end function AboutBox_run

  function AboutDlgProc(hDlg,iMsg,wParam,lParam) bind(C)
    !GCC$ ATTRIBUTES STDCALL :: AboutDlgProc
    integer(BOOL_T) :: AboutDlgProc
    integer(HWND_T), intent(in), value :: hDlg
    integer(UINT_T), intent(in), value :: iMsg
```

```
    integer(WPARAM_T), intent(in), value :: wParam
    integer(LPARAM_T), intent(in), value :: lParam
    integer :: dummy

    ! To avoid some annoying warnings...
    integer(LPARAM_T) :: not_used_lParam
    not_used_lParam = lParam

    select case(iMsg)
    case(WM_INITDIALOG)
       dialog_result = FALSE_T
       AboutDlgProc = TRUE_T
       return

    case(WM_COMMAND)
       select case(lo_word(int(wParam,DWORD_T)))
       case(IDOK)
          dummy = EndDialog(hDlg,int(IDOK,INT_PTR_T))
          dialog_result = TRUE_T
          AboutDlgProc = TRUE_T
          return

       case(IDCANCEL)
          dummy = EndDialog(hDlg,int(IDCANCEL,INT_PTR_T))
          AboutDlgProc = TRUE_T
          return

       end select
    end select

    AboutDlgProc = FALSE_T
    return
  end function AboutDlgProc
end module AboutBox_class

module XBox_class
  use, intrinsic :: iso_c_binding, only: c_funloc
  use kind_consts, only: DP
  use win32, only: BOOL_T, DWORD_T, FALSE_T, HWND_T, IDCANCEL, IDOK, INT_T, &
       INT_PTR_T, LPARAM_T, MAX_FMT, MAX_LEN, MB_ICONINFORMATION, MB_OK, &
       NUL, NULL_LPSTR, TRUE_T, UINT_T, WM_COMMAND, WM_INITDIALOG, WORD_T, &
       WPARAM_T, &
       dialog_box, EndDialog, GetDlgItemText, GetModuleHandle, lo_word, &
       make_int_resource, MessageBox, PostQuitMessage, SetDlgItemText
  implicit none
  private

  type, public :: XBox
     private
     integer(HWND_T) :: hDlg
     integer(WORD_T) :: idd_data
     integer(INT_T) :: idc_x
     character(len=MAX_FMT) :: fmt_str
     real(DP) :: x
  end type XBox

  type(XBox) :: xb

  ! TRUE_T if OK button is pressed, otherwise it is FALSE_T
  integer(BOOL_T) :: dialog_result = FALSE_T

  interface new_box
     module procedure XBox_init
  end interface new_box

  interface run
     module procedure XBox_run
  end interface run

  public :: new_box, run, get

contains

  subroutine XBox_init(this,hhDlg,idd_data_xx,idc_xx,fmt,xx)
    type(XBox), intent(out) :: this
    integer(HWND_T), intent(in) :: hhDlg
    integer(WORD_T), intent(in) :: idd_data_xx
    integer(INT_T), intent(in) :: idc_xx
```

```
  character(len=MAX_FMT), intent(in) :: fmt
  real(DP), intent(in) :: xx

  this%hDlg = hhDlg
  this%idd_data = idd_data_xx
  this%idc_x = idc_xx
  this%fmt_str = fmt
  this%x = xx
end subroutine XBox_init

function XBox_run(this)
  integer(BOOL_T) :: XBox_run
  type(XBox), intent(inout) :: this
  integer(INT_PTR_T) :: dummy

  ! Input
  xb = this

  dummy = dialog_box(GetModuleHandle(NULL_LPSTR), &
       make_int_resource(xb%idd_data),xb%hDlg,c_funloc(XDlgProc))

  ! Output
  this = xb

  XBox_run = dialog_result
end function XBox_run

function get(this)
  real(DP) :: get
  type(XBox), intent(in) :: this
  get = this%x
end function get

function XDlgProc(hDlg,iMsg,wParam,lParam) bind(C)
  !GCC$ ATTRIBUTES STDCALL :: XDlgProc
  integer(BOOL_T) :: XDlgProc
  integer(HWND_T), intent(in), value :: hDlg
  integer(UINT_T), intent(in), value :: iMsg
  integer(WPARAM_T), intent(in), value :: wParam
  integer(LPARAM_T), intent(in), value :: lParam

  ! To avoid some annoying warnings...
  integer(LPARAM_T) :: not_used_lParam
  not_used_lParam = lParam

  select case(iMsg)
  case(WM_INITDIALOG)
     call init_dialog(hDlg)
     dialog_result = FALSE_T
     XDlgProc = TRUE_T
     return

  case(WM_COMMAND)
     select case(lo_word(int(wParam,DWORD_T)))
     case(IDOK)
        call ok_command(hDlg)
        dialog_result = TRUE_T
        XDlgProc = TRUE_T
        return

     case(IDCANCEL)
        call cancel_command(hDlg)
        XDlgProc = TRUE_T
        return

     end select
  end select

  XDlgProc = FALSE_T
  return
end function XDlgProc

subroutine init_dialog(hDlg)
  integer(HWND_T), intent(in) :: hDlg
  character(len=MAX_LEN) :: buffer
  integer :: dummy

  buffer = ''
```

```
    write(buffer,xb%fmt_str) xb%x
    dummy = SetDlgItemText(hDlg,xb%idc_x,trim(adjustl(buffer))//NUL)
  end subroutine init_dialog

  subroutine ok_command(hDlg)
    integer(HWND_T), intent(in) :: hDlg
    character(len=MAX_LEN) :: buffer
    integer :: dummy, ierr
    real(DP) :: x_try = 0.0_DP

    buffer = ''
    dummy = GetDlgItemText(hDlg,xb%idc_x,buffer,MAX_LEN)
    if (dummy > 0) then
       dummy = index(buffer,NUL)
       read(buffer(1:dummy−1),*,iostat = ierr) x_try
       if (ierr /= 0) then
          write(*,*) 'IERR, X = ', ierr, x_try
       else
          xb%x = x_try
       end if
    else
       dummy = MessageBox(hDlg,'Failure reading X data!'//NUL, &
            'Fatal Error!!!'//NUL, &
            ior(MB_OK,MB_ICONINFORMATION))
       call PostQuitMessage(1) ! Exit code 1 to flag an error occured
    end if

    dummy = EndDialog(hDlg,int(IDOK,INT_PTR_T))
  end subroutine ok_command

  subroutine cancel_command(hDlg)
    integer(HWND_T), intent(in) :: hDlg
    integer :: dummy
    dummy = EndDialog(hDlg,int(IDCANCEL,INT_PTR_T))
  end subroutine cancel_command
end module XBox_class

module XYBox_class
  use, intrinsic :: iso_c_binding, only: c_funloc
  use kind_consts, only: DP
  use win32, only: BOOL_T, DWORD_T, FALSE_T, HWND_T, IDCANCEL, IDOK, INT_T, &
       INT_PTR_T, LPARAM_T, MAX_FMT, MAX_LEN, MB_ICONINFORMATION, MB_OK, &
       NUL, NULL_LPSTR, TRUE_T, UINT_T, WM_COMMAND, WM_INITDIALOG, WORD_T, &
       WPARAM_T, &
       dialog_box, EndDialog, GetDlgItemText, GetModuleHandle, lo_word, &
       make_int_resource, MessageBox, PostQuitMessage, SetDlgItemText
  implicit none
  private

  type, public :: XYBox
     private
     integer(HWND_T) :: hDlg
     integer(WORD_T) :: idd_data
     integer(INT_T) :: idc_x, idc_y
     character(len=MAX_FMT) :: fmt_str
     real(DP) :: x, y
  end type XYBox

  type(XYBox) :: xyb

  ! TRUE_T if OK button is pressed, otherwise it is FALSE_T
  integer(BOOL_T) :: dialog_result = FALSE_T

  interface new_box
     module procedure XYBox_init
  end interface new_box

  interface run
     module procedure XYBox_run
  end interface run

  public :: new_box, run, get_x, get_y

contains

  subroutine XYBox_init(this,hhDlg,idd_data_xy,idc_xx,idc_yy,fmt,xx,yy)
    type(XYBox), intent(out) :: this
    integer(HWND_T), intent(in) :: hhDlg
```

```
  integer(WORD_T), intent(in) :: idd_data_xy
  integer(INT_T), intent(in) :: idc_xx, idc_yy
  character(len=MAX_FMT), intent(in) :: fmt
  real(DP), intent(in) :: xx, yy

  this%hDlg = hhDlg
  this%idd_data = idd_data_xy
  this%idc_x = idc_xx
  this%idc_y = idc_yy
  this%fmt_str = fmt
  this%x = xx
  this%y = yy
end subroutine XYBox_init

function XYBox_run(this)
  integer(BOOL_T) :: XYBox_run
  type(XYBox), intent(inout) :: this
  integer(INT_PTR_T) :: dummy

  ! Input
  xyb = this

  dummy = dialog_box(GetModuleHandle(NULL_LPSTR), &
       make_int_resource(xyb%idd_data),xyb%hDlg,c_funloc(XYDlgProc))

  ! Output
  this = xyb

  XYBox_run = dialog_result
end function XYBox_run

function get_x(this)
  real(DP) :: get_x
  type(XYBox), intent(in) :: this
  get_x = this%x
end function get_x

function get_y(this)
  real(DP) :: get_y
  type(XYBox), intent(in) :: this
  get_y = this%y
end function get_y

function XYDlgProc(hDlg,iMsg,wParam,lParam) bind(C)
  !GCC$ ATTRIBUTES STDCALL :: XYDlgProc
  integer(BOOL_T) :: XYDlgProc
  integer(HWND_T), intent(in), value :: hDlg
  integer(UINT_T), intent(in), value :: iMsg
  integer(WPARAM_T), intent(in), value :: wParam
  integer(LPARAM_T), intent(in), value :: lParam

  ! To avoid some annoying warnings...
  integer(LPARAM_T) :: not_used_lParam
  not_used_lParam = lParam

  select case(iMsg)
  case(WM_INITDIALOG)
     call init_dialog(hDlg)
     dialog_result = FALSE_T
     XYDlgProc = TRUE_T
     return

  case(WM_COMMAND)
     select case(lo_word(int(wParam,DWORD_T)))
     case(IDOK)
        call ok_command(hDlg)
        dialog_result = TRUE_T
        XYDlgProc = TRUE_T
        return

     case(IDCANCEL)
        call cancel_command(hDlg)
        XYDlgProc = TRUE_T
        return

     end select
  end select
```

```
    XYDlgProc = FALSE_T
    return
  end function XYDlgProc

  subroutine init_dialog(hDlg)
    integer(HWND_T), intent(in) :: hDlg
    character(len=MAX_LEN) :: buffer
    integer :: dummy

    buffer = ''
    write(buffer,xyb%fmt_str) xyb%x
    dummy = SetDlgItemText(hDlg,xyb%idc_x,trim(adjustl(buffer))//NUL)

    buffer = ''
    write(buffer,xyb%fmt_str) xyb%y
    dummy = SetDlgItemText(hDlg,xyb%idc_y,trim(adjustl(buffer))//NUL)
  end subroutine init_dialog

  subroutine ok_command(hDlg)
    integer(HWND_T), intent(in) :: hDlg
    character(len=MAX_LEN) :: buffer
    integer :: dummy, ierr
    real(DP) :: x_try = 0.0_DP, y_try = 0.0_DP

    buffer = ''
    dummy = GetDlgItemText(hDlg,xyb%idc_x,buffer,MAX_LEN)
    if (dummy > 0) then
       dummy = index(buffer,NUL)
       read(buffer(1:dummy-1),*,iostat = ierr) x_try
       if (ierr /= 0) then
          write(*,*) 'IERR, X = ', ierr, x_try
       else
          xyb%x = x_try
       end if
    else
       dummy = MessageBox(hDlg,'Failure reading X data!'//NUL, &
            'Fatal Error!!!'//NUL, &
            ior(MB_OK,MB_ICONINFORMATION))
       call PostQuitMessage(1) ! Exit code 1 to flag an error occured
    end if

    buffer = ''
    dummy = GetDlgItemText(hDlg,xyb%idc_y,buffer,MAX_LEN)
    if (dummy > 0) then
       dummy = index(buffer,NUL)
       read(buffer(1:dummy-1),*,iostat = ierr) y_try
       if (ierr /= 0) then
          write(*,*) 'IERR, Y = ', ierr, y_try
       else
          xyb%y = y_try
       end if
    else
       dummy = MessageBox(hDlg,'Failure reading Y data!'//NUL, &
            'Fatal Error!!!'//NUL, &
            ior(MB_OK,MB_ICONINFORMATION))
       call PostQuitMessage(1) ! Exit code 1 to flag an error occured
    end if

    dummy = EndDialog(hDlg,int(IDOK,INT_PTR_T))
  end subroutine ok_command

  subroutine cancel_command(hDlg)
    integer(HWND_T), intent(in) :: hDlg
    integer :: dummy
    dummy = EndDialog(hDlg,int(IDCANCEL,INT_PTR_T))
  end subroutine cancel_command
end module XYBox_class

module RadioBox_class
  use, intrinsic :: iso_c_binding, only: c_funloc
  use win32, only: BOOL_T, DWORD_T, FALSE_T, HWND_T, IDCANCEL, IDOK, INT_T, &
       INT_PTR_T, LPARAM_T, MAX_LEN, NUL, NULL_LPSTR, TRUE_T, UINT_T, &
       WM_COMMAND, WM_INITDIALOG, WORD_T, WPARAM_T, &
       CheckRadioButton, dialog_box, EndDialog, GetModuleHandle, lo_word, &
       make_int_resource, SetDlgItemText
  implicit none
  private
```

```
  integer, parameter :: MAX_RADIO_BUTTONS = 10

  type, public :: RadioBox
     private
     integer(HWND_T) :: hDlg
     integer(WORD_T) :: idd_radio
     integer(INT_T) :: idc_first_button, idc_last_button, idc_current_button
     character(len=MAX_LEN) :: button_names(MAX_RADIO_BUTTONS)
     integer :: num_buttons
     integer :: current_button
  end type RadioBox

  type(RadioBox) :: rb

  ! TRUE_T if OK button is pressed, otherwise it is FALSE_T
  integer(BOOL_T) :: dialog_result = FALSE_T

  interface new_box
     module procedure RadioBox_init
  end interface new_box

  interface run
     module procedure RadioBox_run
  end interface run

  public :: new_box, run, get_current_button

contains

  subroutine RadioBox_init(this,hDlg,idd_radio,idc_first_button, &
       button_names,num_buttons,current_button)
    type(RadioBox), intent(out) :: this
    integer(HWND_T), intent(in) :: hDlg
    integer(WORD_T), intent(in) :: idd_radio
    integer(INT_T), intent(in) :: idc_first_button
    character(len=*), intent(in) :: button_names(:)
    integer, intent(in) :: num_buttons
    integer, intent(in) :: current_button
    integer :: i

    if (num_buttons > MAX_RADIO_BUTTONS) then
       write(*,*) '*** FATAL ERROR ***'
       write(*,*) 'NUM_BUTTONS > ', MAX_RADIO_BUTTONS, ' NOT ALLOWED!!!'
       write(*,*) 'Program terminates...'
       stop
    end if

    this%hDlg = hDlg
    this%idd_radio = idd_radio
    this%idc_first_button = idc_first_button
    this%idc_last_button = idc_first_button+(num_buttons−1)

    do i = 1, num_buttons
       this%button_names(i) = trim(adjustl(button_names(i)))
    end do

    this%num_buttons = num_buttons
    this%current_button = current_button
  end subroutine RadioBox_init

  function RadioBox_run(this)
    integer(BOOL_T) :: RadioBox_run
    type(RadioBox), intent(inout) :: this
    integer(INT_PTR_T) :: dummy

    ! Input
    rb = this

    dummy = dialog_box(GetModuleHandle(NULL_LPSTR), &
         make_int_resource(rb%idd_radio),rb%hDlg,c_funloc(RadioDlgProc))

    ! Output
    this = rb

    RadioBox_run = dialog_result
  end function RadioBox_run

  function get_current_button(this)
```

```
  integer :: get_current_button
  type(RadioBox), intent(in) :: this
  get_current_button = this%current_button
end function get_current_button

function RadioDlgProc(hDlg,iMsg,wParam,lParam) bind(C)
  !GCC$ ATTRIBUTES STDCALL :: RadioDlgProc
  integer(BOOL_T) :: RadioDlgProc
  integer(HWND_T), intent(in), value :: hDlg
  integer(UINT_T), intent(in), value :: iMsg
  integer(WPARAM_T), intent(in), value :: wParam
  integer(LPARAM_T), intent(in), value :: lParam
  integer :: dummy

  ! To avoid some annoying warnings...
  integer(LPARAM_T) :: not_used_lParam
  not_used_lParam = lParam

  ! Now we use dummy to store the current button if it is valid. Se below...
  dummy = lo_word(int(wParam,DWORD_T))

  select case(iMsg)
  case(WM_INITDIALOG)
     call init_dialog(hDlg)
     dialog_result = FALSE_T
     RadioDlgProc = TRUE_T
     return

  case(WM_COMMAND)
     ! We test if the current button is valid...
     if ((rb%idc_first_button <= dummy) .and. &
          (dummy <= rb%idc_last_button)) then

        ! ...being valid, we save it...
        rb%idc_current_button = dummy

        ! Now dumy is "free" and can be reused... :−)
        dummy = CheckRadioButton(hDlg, &
             rb%idc_first_button,rb%idc_last_button,rb%idc_current_button)
        RadioDlgProc = TRUE_T
        return

     end if

     ! ...if it is not valid, it could be something else...
     select case(dummy)
     case(IDOK)
        call ok_command(hDlg)
        dialog_result = TRUE_T
        RadioDlgProc = TRUE_T
        return

     case(IDCANCEL)
        call cancel_command(hDlg)
        RadioDlgProc = TRUE_T
        return

     end select
  end select

  RadioDlgProc = FALSE_T
  return
end function RadioDlgProc

subroutine init_dialog(hDlg)
  integer(HWND_T), intent(in) :: hDlg
  integer :: i, dummy

  do i = 1, rb%num_buttons
     dummy = SetDlgItemText(hDlg,rb%idc_first_button+(i−1), &
          trim(adjustl(rb%button_names(i)))//NUL)
  end do

  rb%idc_current_button = rb%idc_first_button+(rb%current_button−1)
  dummy = CheckRadioButton(hDlg, &
       rb%idc_first_button,rb%idc_last_button,rb%idc_current_button)
end subroutine init_dialog
```

```
  subroutine ok_command(hDlg)
    integer(HWND_T), intent(in) :: hDlg
    integer :: dummy
    rb%current_button = (rb%idc_current_button−rb%idc_first_button)+1
    dummy = EndDialog(hDlg,int(IDOK,INT_PTR_T))
  end subroutine ok_command

  subroutine cancel_command(hDlg)
    integer(HWND_T), intent(in) :: hDlg
    integer :: dummy
    dummy = EndDialog(hDlg,int(IDCANCEL,INT_PTR_T))
  end subroutine cancel_command
end module RadioBox_class
```

```
!
! (Partial) Fortran Interface to the Windows API Library
! by Angelo Graziosi  (firstname.lastnameATalice.it)
! Copyright Angelo Graziosi
!
! It is distributed in the hope that it will be useful,
! but WITHOUT ANY WARRANTY; without even the implied warranty of
! MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
!
! DESCRIPTION
!   win32app module
!   Just to start with Windows applications in World Coordinete System...
!
!   Notice that:
!
!       int(0,UINT_T)    -->  0_UINT_T
!       int(0,WPARAM_T)  -->  0_WPARAM_T
!       int(0,LPARAM_T)  -->  0_LPARAM_T
!       ...
!

module win32app
  use kind_consts, only: DP
  use win32, only: BOOL_T, DWORD_T, HBITMAP_T, HBRUSH_T, HDC_T, HWND_T, INT_T, &
       LPARAM_T, MAX_FMT, NUL, SRCCOPY, WORD_T, &
       RECT_T, &
       BitBlt, CreateCompatibleBitmap, Ellipse, FillRect, hi_word, lo_word, &
       Rectangle, TextOut
  use XYBox_class
  implicit none
  private

  character(len=MAX_FMT), parameter :: FMT = '(1pg12.5)'

  integer  :: client_width = 0, client_height = 0

  type, public :: box_type
     real(DP) :: x1, x2
     real(DP) :: y1, y2
  end type box_type

  ! Output view region in WC
  real(DP) :: x_min = −1.0_DP, x_max = 1.0_DP, &
       y_min = −1.0_DP, y_max = 1.0_DP, &
       dx = 1.0_DP, dy = 1.0_DP

  public :: win32app_BitBlt, win32app_clearDC, &
       win32app_CreateCompatibleBitmap, win32app_ellipse, win32app_fillbox, &
       win32app_setup, win32app_textout, &
       win32app_xbounds, win32app_ybounds, &
       win32app_xmin, win32app_xmax, win32app_ymin, win32app_ymax, &
       win32app_height, win32app_width

contains

  subroutine win32app_setup(lParam,x1,x2,y1,y2)
    integer(LPARAM_T), intent(in) :: lParam
    real(DP), intent(in), optional :: x1, x2, y1, y2
    real(DP) :: cx, cy

    ! Initializing with defaults values...
    if (present(x1)) x_min = x1
    if (present(x2)) x_max = x2
    if (present(y1)) y_min = y1
    if (present(y2)) y_max = y2

    ! The true width and height of client area
    client_width = lo_word(int(lParam,DWORD_T)) + 1
    client_height = hi_word(int(lParam,DWORD_T)) + 1

    dx = x_max−x_min
    dy = y_max−y_min

    cx = x_min+0.5_DP*dx
    cy = y_min+0.5_DP*dy

    ! First, adjusts WC region...
    if (client_width > client_height) then
```

```
      dy = (dx*client_height)/client_width
      y_min = cy-0.5_DP*dy
      y_max = y_min+dy
   else
      dx = (dy*client_width)/client_height
      x_min = cx-0.5_DP*dx
      x_max = x_min+dx
   end if

   ! ...then, calculates the size of the mesh that represents each pixel
   dx = (x_max-x_min)/client_width
   dy = (y_max-y_min)/client_height

   ! Many Windows routines expect a "virtual" width and height,
   ! more precisely the client area bottom-right point coordinates
   client_width = client_width - 1
   client_height = client_height - 1
end subroutine win32app_setup

function xs(x)
   integer :: xs
   real(DP), intent(in) :: x
   xs = 0+int((x-x_min)/dx)
end function xs

function ys(y)
   integer :: ys
   real(DP), intent(in) :: y
   ys = 0+int((y_max-y)/dy)
end function ys

function win32app_xmin() result(r)
   real(DP) :: r
   r = x_min
end function win32app_xmin

function win32app_xmax() result(r)
   real(DP) :: r
   r = x_max
end function win32app_xmax

function win32app_ymin() result(r)
   real(DP) :: r
   r = y_min
end function win32app_ymin

function win32app_ymax() result(r)
   real(DP) :: r
   r = y_max
end function win32app_ymax

function win32app_width() result(r)
   integer :: r
   r = client_width !+1 ?
end function win32app_width

function win32app_height() result(r)
   integer :: r
   r = client_height !+1 ?
end function win32app_height

subroutine win32app_xbounds(hWnd,idd_data_xlimits,idc_xmin,idc_xmax)
   integer(HWND_T), intent(in) :: hWnd
   integer(WORD_T), intent(in) :: idd_data_xlimits
   integer(INT_T), intent(in) :: idc_xmin, idc_xmax
   type(XYBox) :: xyb
   real(DP) :: u_min, u_max, du, c_params

   ! The current Y view center
   c_params = 0.5_DP*(y_max+y_min)

   call new_box(xyb,hWnd,idd_data_xlimits,idc_xmin,idc_xmax,FMT,x_min,x_max)

   if (run(xyb) > 0) then
      u_min = get_x(xyb)
      u_max = get_y(xyb)

      ! The assumed new intervall size
```

```
      du = u_max−u_min

      ! If it is too small
      if (abs(du) <= 0.0_DP) then
         u_min = −2.0_DP
         u_max = 2.0_DP
         du = 4.0_DP
      else
         ! if u_max < u_min
         if (du < 0.0_DP) then
            ! swap u_min/max using du as temp
            du = u_max
            u_max = u_min
            u_min = du
            du = u_max−u_min
         end if
      end if

      ! The new Y height (with the same aspect ratio)
      du = du*(y_max−y_min)/(x_max−x_min)

      ! The X limits just inserted
      x_min = u_min
      x_max = u_max

      ! Adjusting the Y limits accordingly
      y_min = c_params−0.5_DP*du
      y_max = y_min+du

      ! We need to recompute the size of the mesh that represents each pixel
      dx = (x_max−x_min)/(client_width+1)
      dy = (y_max−y_min)/(client_height+1)
   end if
end subroutine win32app_xbounds

subroutine win32app_ybounds(hWnd,idd_data_ylimits,idc_ymin,idc_ymax)
   integer(HWND_T), intent(in) :: hWnd
   integer(WORD_T), intent(in) :: idd_data_ylimits
   integer(INT_T), intent(in) :: idc_ymin, idc_ymax
   type(XYBox) :: xyb
   real(DP) :: u_min, u_max, du, c_params

   ! The current X view center
   c_params = 0.5_DP*(x_max+x_min)

   call new_box(xyb,hWnd,idd_data_ylimits,idc_ymin,idc_ymax,FMT,y_min,y_max)

   if (run(xyb) > 0) then
      u_min = get_x(xyb)
      u_max = get_y(xyb)

      ! The assumed new intervall size
      du = u_max−u_min

      ! If it is too small
      if (abs(du) <= 0.0_DP) then
         u_min = −2.0_DP
         u_max = 2.0_DP
         du = 4.0_DP
      else
         ! if u_max < u_min
         if (du < 0.0_DP) then
            ! swap u_min/max using du as temp
            du = u_max
            u_max = u_min
            u_min = du
            du = u_max−u_min
         end if
      end if

      ! The new X width (with the same aspect ratio)
      du = du*(x_max−x_min)/(y_max−y_min)

      ! The Y limits just inserted
      y_min = u_min
      y_max = u_max

      ! Adjusting the X limits accordingly
```

```
      x_min = c_params−0.5_DP*du
      x_max = x_min+du

      ! We need to recompute the size of the mesh that represents each pixel
      dx = (x_max−x_min)/(client_width+1)
      dy = (y_max−y_min)/(client_height+1)
    end if
  end subroutine win32app_ybounds

  function win32app_BitBlt(hdc,hdcMem) result(r)
    integer(BOOL_T) :: r
    integer(HDC_T), intent(in) :: hdc, hdcMem
    r = BitBlt(hdc,0,0,client_width,client_height,hdcMem,0,0,SRCCOPY)
  end function win32app_BitBlt

  function win32app_clearDC(hdc,dwRop) result(r)
    integer(BOOL_T) :: r
    integer(HDC_T), intent(in) :: hdc
    integer(DWORD_T), intent(in) :: dwRop
    ! dwRop = BLACKNESS or WHITENESS?
    r = BitBlt(hdc,0,0,client_width,client_height,0_HDC_T,0,0,dwRop)
    !r = Rectangle(hdc,−1,−1,client_width+1,client_height+1)
  end function win32app_clearDC

  function win32app_CreateCompatibleBitmap(hdc) result(r)
    integer(HBITMAP_T) :: r
    integer(HDC_T), intent(in) :: hdc
    r = CreateCompatibleBitmap(hdc,client_width,client_height)
  end function win32app_CreateCompatibleBitmap

  function win32app_ellipse(hdc,left,top,right,bottom) result(r)
    integer(BOOL_T) :: r
    integer(HDC_T), intent(in) :: hdc
    real(DP), intent(in) :: left,top,right,bottom
    r = Ellipse(hdc,xs(left),ys(top),xs(right),ys(bottom))
  end function win32app_ellipse

  function win32app_fillbox(hdc,box,hBrush) result(r)
    integer(INT_T) :: r
    integer(HDC_T), intent(in) :: hdc
    type(box_type), intent(in) :: box
    integer(HBRUSH_T), intent(in) :: hBrush
    type(RECT_T), save :: rect

    rect%left = xs(box%x1)
    rect%right = xs(box%x2)
    rect%bottom = ys(box%y1)
    rect%top = ys(box%y2)

    r = FillRect(hdc,rect,hBrush)
  end function win32app_fillbox

  function win32app_textout(hdc,x,y,text) result(r)
    integer(INT_T) :: r
    integer(HDC_T), intent(in) :: hdc
    real(DP), intent(in) :: x, y
    character(len=*), intent(in) :: text

    r = index(text,NUL)
    r = TextOut(hdc,xs(x),ys(y),text(1:r),r−1)
  end function win32app_textout
end module win32app
```

```
!
! (Partial) Fortran Interface to the Windows API Library
! by Angelo Graziosi  (firstname.lastnameATalice.it)
! Copyright Angelo Graziosi
!
! It is distributed in the hope that it will be useful,
! but WITHOUT ANY WARRANTY; without even the implied warranty of
! MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
!
! HOW TO BUILD (MSYS2/MINGW32/MINGW64 shell)
!
!   cd ~/programming/win32−fortran/bounce
!
!   rm −rf {*.mod,*.res,~/programming/modules/*} && \
!   windres bounce.rc −O coff −o bounce.res && \
!   gfortran −O3 −Wall −mwindows −J ~/programming/modules \
!     ~/programming/basic−modules/basic_mods.f90 \
!     ../{win32.f90,win32boxes.f90,win32app.f90} bounce.f90 \
!     bounce.res −o bounce.out && \
!     rm −rf {*.mod,*.res,~/programming/modules/*}
!
! In MINGW32/MINGW64, add '−static' and:
!
!   bounce.out ==> bounce−mingw32/mingw64
!
!
! Remember that:
!
!   int(0,UINT_T)    −−>  0_UINT_T
!   int(0,WPARAM_T)  −−>  0_WPARAM_T
!   int(0,LPARAM_T)  −−>  0_LPARAM_T
!   ...
!

module the_app
  use kind_consts, only: DP
  use AboutBox_class
  use XBox_class
  use XYBox_class
  use win32, only: BLACK_COLOR, BOOL_T, DWORD_T, FALSE_T, HBITMAP_T, &
       HBRUSH_T, HDC_T, HINSTANCE_T, HS_DIAGCROSS, HWND_T, IDYES, INT_T, &
       LPARAM_T, LRESULT_T, MAX_FMT, MAX_LEN, NL, NUL, NULL_T, TRUE_T, &
       UINT_T, WHITENESS, WM_CLOSE, WM_COMMAND, WM_DESTROY, WM_SIZE, WORD_T, &
       WPARAM_T, &
       ask_confirmation, CreateCompatibleDC, CreateHatchBrush, &
       DefWindowProc, DeleteDC, DeleteObject, DestroyWindow, error_msg, &
       GetDC, lo_word, MessageBeep, PostMessage, PostQuitMessage, ReleaseDC, &
       RGB, SelectObject, SetBkColor, TextOut
  use win32app, only: win32app_BitBlt, win32app_clearDC, &
       win32app_CreateCompatibleBitmap, win32app_ellipse, win32app_setup, &
       win32app_xbounds, win32app_ybounds, &
       win32app_xmin, win32app_xmax, win32app_ymin, win32app_ymax
  implicit none
  private

  integer(WORD_T), parameter, public :: IDI_BOUNCE = 1
  integer(WORD_T), parameter, public :: IDM_MAINMENU = 9000

  integer(WORD_T), parameter :: IDM_FILE_EXIT    = 9010
  integer(WORD_T), parameter :: IDM_DATA_RADIUS  = 9020
  integer(WORD_T), parameter :: IDM_DATA_SPEED   = 9021
  integer(WORD_T), parameter :: IDM_DATA_TTOT    = 9022
  integer(WORD_T), parameter :: IDM_DATA_TSTEP   = 9023
  integer(WORD_T), parameter :: IDM_DATA_XBOUNDS = 9024
  integer(WORD_T), parameter :: IDM_DATA_YBOUNDS = 9025
  integer(WORD_T), parameter :: IDM_RUNAPP       = 9030
  integer(WORD_T), parameter :: IDM_HELP_ABOUT   = 9999

  !integer(WORD_T), parameter :: IDC_STATIC = −1

  integer(WORD_T), parameter :: IDD_DATA_RADIUS = 100
  integer(INT_T), parameter :: IDC_RADIUS       = 101

  integer(WORD_T), parameter :: IDD_DATA_SPEED = 150
  integer(INT_T), parameter :: IDC_SPEED       = 151

  integer(WORD_T), parameter :: IDD_DATA_TTOT = 200
  integer(INT_T), parameter :: IDC_TMIN       = 201
```

```
  integer(INT_T), parameter :: IDC_TMAX       = 202

  integer(WORD_T), parameter :: IDD_DATA_TSTEP = 300
  integer(INT_T), parameter :: IDC_TSTEP       = 301

  integer(WORD_T), parameter :: IDD_DATA_XBOUNDS = 400
  integer(INT_T), parameter :: IDC_XMIN          = 401
  integer(INT_T), parameter :: IDC_XMAX          = 402

  integer(WORD_T), parameter :: IDD_DATA_YBOUNDS = 500
  integer(INT_T), parameter :: IDC_YMIN          = 501
  integer(INT_T), parameter :: IDC_YMAX          = 502

  integer(WORD_T), parameter :: IDD_ABOUT = 999

  ! COMMON data
  integer(HBITMAP_T) :: hBitmap = NULL_T
  logical :: run_flag = .true.
  real(DP) :: box_xmin, box_xmax, box_ymin, box_ymax

  ! Application data, strictly speaking...
  real(DP) :: p(2) = 0.0_DP, v(2) = 0.0_DP, radius = 10.0_DP, speed = 10.0_DP

  real(DP) :: t0 = 0.0_DP, t1 = 900.0_DP, &
       tstep = 1.0_DP/16 ! 0.0625 = 0.0001_2

  real(DP) :: t = 0.0_DP

  public :: paint_screen, WndProc

contains

  subroutine setup_ball()
    use math_consts, only: DEG2RAD, PI
    real(DP) :: u, phi

    ! Time initialization
    t = t0

    ! The initial ball position (of its center)
    p = [ 0.5_DP*(box_xmin+box_xmax), 0.5_DP*(box_ymin+box_ymax) ]

    ! Initial moving (random) direction
    call random_number(u)

    phi = (u*360.0_DP)*DEG2RAD
    v = speed*[ cos(phi), sin(phi) ]
  end subroutine setup_ball

  subroutine draw_ball(hdc,t)
    integer(HDC_T), intent(in) :: hdc
    real(DP), intent(in) :: t
    ! We use SAVE just to save something at each call
    ! (draw_ball() is called intensively, at each iteration)
    character(len=MAX_LEN), save :: buffer = ''
    integer(HBRUSH_T), save :: hBrush = NULL_T
    integer, save :: dummy

    buffer = ''
    write(buffer,*) 'Time : ',t
    buffer = trim(adjustl(buffer))//'   '//NUL
    dummy = index(buffer,NUL)
    dummy = TextOut(hdc,0,0,buffer(1:dummy),dummy−1)

    hBrush = CreateHatchBrush(HS_DIAGCROSS,BLACK_COLOR)
    !hBrush = CreateHatchBrush(HS_DIAGCROSS,YELLOW_COLOR)

    dummy = int(SelectObject(hdc,hBrush),INT_T)
    dummy = SetBkColor(hdc,RGB(255,0,255))
    dummy = win32app_ellipse(hdc,p(1)−radius,p(2)+radius, &
         p(1)+radius,p(2)−radius)
    dummy = DeleteObject(hBrush)
  end subroutine draw_ball

  subroutine painting_setup(hWnd)
    integer(HWND_T), intent(in) :: hWnd

    logical, save :: first = .true.
```

```
  integer(HDC_T) :: hdc, hdcMem
  integer :: dummy

  if (first) then
     call setup_ball()
     first = .false.
  end if

  if (hBitmap /= NULL_T) then
     dummy = DeleteObject(hBitmap)
  end if

  hdc = GetDC(hWnd)
  hdcMem = CreateCompatibleDC(hdc)
  hBitmap = win32app_CreateCompatibleBitmap(hdc)
  dummy = ReleaseDC(hWnd,hdc)

  dummy = int(SelectObject(hdcMem,hBitmap),INT_T)

  ! Clear the off−screen DC (hdcMem) for the next drawing
  dummy = win32app_clearDC(hdcMem,WHITENESS)

  call draw_ball(hdcMem,t)

  dummy = DeleteDC(hdcMem)
end subroutine painting_setup

subroutine set_radius(hWnd)
  integer(HWND_T), intent(in) :: hWnd
  character(len=MAX_FMT), parameter :: FMT = '(1pg12.5)'
  type(XBox) :: xb

  call new_box(xb,hWnd,IDD_DATA_RADIUS,IDC_RADIUS,FMT,radius)

  if (run(xb) > 0) then
     radius = get(xb)

     if (radius < 0) then
        call error_msg('Radius < 0 !!!'//NL &
             //'Taking its absolute value...   '//NUL)
        radius = abs(radius)
     end if
  end if
end subroutine set_radius

subroutine set_speed(hWnd)
  integer(HWND_T), intent(in) :: hWnd
  character(len=MAX_FMT), parameter :: FMT = '(1pg12.5)'
  type(XBox) :: xb

  call new_box(xb,hWnd,IDD_DATA_SPEED,IDC_SPEED,FMT,speed)

  if (run(xb) > 0) then
     speed = get(xb)

     if (speed < 0) then
        call error_msg('Speed < 0 !!!'//NL &
             //'Taking its absolute value...   '//NUL)
        speed = abs(speed)
     end if
  end if
end subroutine set_speed

subroutine set_timebounds(hWnd)
  integer(HWND_T), intent(in) :: hWnd
  character(len=MAX_FMT), parameter :: FMT = '(1pg12.5)'
  type(XYBox) :: xyb

  call new_box(xyb,hWnd,IDD_DATA_TTOT,IDC_TMIN,IDC_TMAX,FMT,t0,t1)

  if (run(xyb) > 0) then
     t0  = min(get_x(xyb),get_y(xyb))
     t1  = max(get_x(xyb),get_y(xyb))
  end if
end subroutine set_timebounds

subroutine set_tstep(hWnd)
  integer(HWND_T), intent(in) :: hWnd
```

```
  character(len=MAX_FMT), parameter :: FMT = '(1pg12.5)'
  type(XBox) :: xb

  call new_box(xb,hWnd,IDD_DATA_TSTEP,IDC_TSTEP,FMT,tstep)

  if (run(xb) > 0) then
     tstep  = get(xb)

     if (tstep < 0) then
        call error_msg('TStep < 0 !!!'//NL &
             //'Taking its absolute value...   '//NUL)
        tstep = abs(tstep)
     end if
  end if
end subroutine set_tstep

subroutine help_dlg(hWnd)
  integer(HWND_T), intent(in) :: hWnd
  type(AboutBox) :: ab
  integer :: dummy
  call new_box(ab,hWnd,IDD_ABOUT)
  dummy = run(ab)
end subroutine help_dlg

function process_command(hWnd,wParam)
  integer(BOOL_T) :: process_command
  integer(HWND_T), intent(in) :: hWnd
  integer(WPARAM_T), intent(in) :: wParam
  integer :: dummy

  run_flag = .false.

  select case(lo_word(int(wParam,DWORD_T)))
  case(IDM_FILE_EXIT)
     dummy = MessageBeep(64)
     if (ask_confirmation(hWnd,'Sure you want to exit?   '//NUL, &
          'Exit?'//NUL) == IDYES) then
        dummy = PostMessage(hWnd,WM_CLOSE,0_WPARAM_T,0_LPARAM_T)
     end if
     process_command = TRUE_T
     return

  case(IDM_DATA_RADIUS)
     call set_radius(hWnd)
     process_command = TRUE_T
     return

  case(IDM_DATA_SPEED)
     call set_speed(hWnd)
     process_command = TRUE_T
     return

  case(IDM_DATA_TTOT)
     call set_timebounds(hWnd)
     process_command = TRUE_T
     return

  case(IDM_DATA_TSTEP)
     call set_tstep(hWnd)
     process_command = TRUE_T
     return

  case(IDM_DATA_XBOUNDS)
     call win32app_xbounds(hWnd,IDD_DATA_XBOUNDS,IDC_XMIN,IDC_XMAX)
     process_command = TRUE_T
     return

  case(IDM_DATA_YBOUNDS)
     call win32app_ybounds(hWnd,IDD_DATA_YBOUNDS,IDC_YMIN,IDC_YMAX)
     process_command = TRUE_T
     return

  case(IDM_RUNAPP)
     run_flag = .true.
     call setup_ball()
     process_command = TRUE_T
     return
```

```
   case(IDM_HELP_ABOUT)
      call help_dlg(hWnd)
      process_command = TRUE_T
      return

   case default
      process_command = FALSE_T
      return
   end select
end function process_command

function WndProc(hWnd,iMsg,wParam,lParam) bind(C)
   !GCC$ ATTRIBUTES STDCALL :: WndProc
   integer(LRESULT_T) :: WndProc
   integer(HWND_T), value :: hWnd
   integer(UINT_T), value :: iMsg
   integer(WPARAM_T), value :: wParam
   integer(LPARAM_T), value :: lParam

   logical, save :: first = .true.
   integer :: dummy

   select case(iMsg)
   case(WM_SIZE)
      if (first) then
         call win32app_setup(lParam,−300.0_DP,300.0_DP)
         first = .false.
      else
         call win32app_setup(lParam)
      end if

      ! Getting the box boundaries... each time, maybe, the mapping changed...
      box_xmin = win32app_xmin()
      box_xmax = win32app_xmax()
      box_ymin = win32app_ymin()
      box_ymax = win32app_ymax()

      ! Now that the mapping has been defined, we can initialize the painting
      call painting_setup(hWnd)

      WndProc = 0
      return

   case(WM_COMMAND)
      if (process_command(hWnd,wParam) == TRUE_T) then
         WndProc = 0
         return
      end if
      ! ...else it continues with DefWindowProc

   case(WM_CLOSE)
      dummy = DestroyWindow(hWnd)
      WndProc = 0
      return

   case(WM_DESTROY)
      if (hBitmap /= NULL_T) then
         dummy = DeleteObject(hBitmap)
      end if

      call PostQuitMessage(0)
      ! Commenting out the next two statements, it continues
      ! with DefWindowProc()
      WndProc = 0
      return
   end select

   WndProc = DefWindowProc(hWnd,iMsg,wParam,lParam)
end function WndProc

subroutine update_ball_position()
   integer, save :: dummy

   ! Computing position at current time. Trial position...
   p = p+v*tstep

   ! ...and correction to keep balls inside the box
```

```
    ! Right
    if (p(1) > box_xmax−radius) then
       v(1) = 0.0_DP−v(1)
       p(1) = box_xmax−radius
       !print *, C_ALERT
       dummy = MessageBeep(0)
    end if

    ! Left
    if (p(1) < box_xmin+radius) then
       v(1) = 0.0_DP−v(1)
       p(1) = box_xmin+radius
       !print *, C_ALERT
       dummy = MessageBeep(0)
    end if

    ! bottom
    if (p(2) < box_ymin+radius) then
       v(2) = 0.0_DP−v(2)
       p(2) = box_ymin+radius
       !print *, C_ALERT
       dummy = MessageBeep(0)
    end if

    ! top
    if (p(2) > box_ymax−radius) then
       v(2) = 0.0_DP−v(2)
       p(2) = box_ymax−radius
       !print *, C_ALERT
       dummy = MessageBeep(0)
    end if
  end subroutine update_ball_position

  subroutine paint_screen(hWnd)
    integer(HWND_T), intent(in) :: hWnd
    ! We use SAVE just to save something at each call
    ! (paint_screen() is called intensively, at each iteration)
    integer(HDC_T), save :: hdc, hdcMem
    integer, save :: dummy

    if (hBitmap /= NULL_T) then
       hdc = GetDC(hWnd)
       hdcMem = CreateCompatibleDC(hdc)
       dummy = int(SelectObject(hdcMem,hBitmap),INT_T)

       ! Transfer the off−screen DC to the screen
       dummy = win32app_BitBlt(hdc,hdcMem)
       dummy = ReleaseDC(hWnd,hdc)

       if (t < t1 .and. run_flag) then
          t = t+tstep
          call update_ball_position()
       end if

       ! Clear the off−screen DC (hdcMem) for the next drawing
       dummy = win32app_clearDC(hdcMem,WHITENESS)

       call draw_ball(hdcMem,t)

       dummy = DeleteDC(hdcMem)
    end if
  end subroutine paint_screen
end module the_app

function WinMain(hInstance,hPrevInstance,lpCmdLine,nCmdShow) &
     bind(C, name='WinMain')
  use randoms, only: init_random_seed
  use, intrinsic :: iso_c_binding, only: C_PTR, C_CHAR, c_sizeof, c_funloc, &
       C_FUNPTR, c_loc
  use win32, only: CS_HREDRAW, CS_VREDRAW, CW_USEDEFAULT, DWORD_T, &
       HINSTANCE_T, HWND_T, INT_T, NUL, NULL_PTR_T, NULL_T, PM_REMOVE, &
       WM_QUIT, WS_OVERLAPPEDWINDOW, UINT_T, WHITE_BRUSH, &
       MSG_T, WNDCLASSEX_T, &
       arrow_cursor, CreateWindowEx, DispatchMessage, error_msg, ExitProcess, &
       GetStockObject, LoadCursor, LoadIcon, make_int_resource, &
       make_int_resource_C_PTR, PeekMessage, RegisterClassEx, ShowWindow, &
       TranslateMessage, UpdateWindow
  use the_app, only: IDI_BOUNCE, IDM_MAINMENU, &
```

```
    paint_screen,  WndProc
implicit none
!GCC$ ATTRIBUTES STDCALL :: WinMain
integer(INT_T) :: WinMain
integer(HINSTANCE_T), value :: hInstance
integer(HINSTANCE_T), value :: hPrevInstance
type(C_PTR), value :: lpCmdLine ! LPSTR
integer(INT_T), value :: nCmdShow

character(kind=C_CHAR,len=128), target :: app_name = &
     'Bounce'//NUL
character(kind=C_CHAR,len=*), parameter :: WINDOW_CAPTION = &
     'Bouncing Ball'//NUL
type(WNDCLASSEX_T) :: WndClass
integer(HWND_T) :: hWnd
type(MSG_T) :: msg
integer :: dummy

! To avoid some annoying warnings...
integer(HINSTANCE_T) :: not_used_hPrevInstance
type(C_PTR) :: not_used_lpCmdLine
not_used_hPrevInstance = hPrevInstance
not_used_lpCmdLine = lpCmdLine

call init_random_seed()

WndClass%cbSize = int(c_sizeof(Wndclass),UINT_T)
WndClass%style = ior(CS_HREDRAW,CS_VREDRAW)
WndClass%lpfnWndProc = c_funloc(WndProc)
WndClass%cbClsExtra = 0
WndClass%cbWndExtra = 0
WndClass%hInstance = hInstance
WndClass%hIcon = LoadIcon(hInstance,make_int_resource(IDI_BOUNCE))
WndClass%hCursor = LoadCursor(NULL_T,arrow_cursor())
WndClass%hbrBackground = GetStockObject(WHITE_BRUSH)
!WndClass%hbrBackground = GetStockObject(BLACK_BRUSH)
WndClass%lpszMenuName = make_int_resource_C_PTR(IDM_MAINMENU)
WndClass%lpszClassName = c_loc(app_name(1:1))
WndClass%hIconSm = LoadIcon(hInstance,make_int_resource(IDI_BOUNCE))

if (RegisterClassEx(WndClass) == 0) then
   call error_msg('Window Registration Failure!   '//NUL)
   call ExitProcess(0_UINT_T)
   WinMain = 0
   !return
end if

hWnd = CreateWindowEx(0_DWORD_T, &
     app_name, &
     WINDOW_CAPTION, &
     WS_OVERLAPPEDWINDOW, &
     CW_USEDEFAULT,CW_USEDEFAULT,CW_USEDEFAULT,CW_USEDEFAULT, &
     NULL_T,NULL_T,hInstance,NULL_PTR_T)

if (hWnd == NULL_T) then
   call error_msg('Window Creation Failure!   '//NUL)
   call ExitProcess(0_UINT_T)
   WinMain = 0
   !return
end if

dummy = ShowWindow(hWnd,nCmdShow)
dummy = UpdateWindow(hWnd)

! See: Charles Petzold "Programming Windows", 5th ed., pag. 162
! 'Random Rectangles'
do
   if (PeekMessage(msg,NULL_T,0,0,PM_REMOVE) /= 0) then
      if (msg%message == WM_QUIT) exit
      dummy = TranslateMessage(msg)
      dummy = int(DispatchMessage(msg),INT_T)
   else
      call paint_screen(hWnd)
   end if
end do

call ExitProcess(int(msg%wParam,UINT_T))
WinMain = 0
```

```
end function WinMain
```

```
//
// (Partial) Fortran Interface to the Windows API Library
// by Angelo Graziosi  (firstname.lastnameATalice.it)
// Copyright Angelo Graziosi
//
// It is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
//
// RC file for "bounce" app
//

#define IDI_BOUNCE 1

IDI_BOUNCE ICON DISCARDABLE "../common_icons/smiling_sun.ico"

#define IDM_MAINMENU      9000
#define IDM_FILE_EXIT     9010
#define IDM_DATA_RADIUS   9020
#define IDM_DATA_SPEED    9021
#define IDM_DATA_TTOT     9022
#define IDM_DATA_TSTEP    9023
#define IDM_DATA_XBOUNDS  9024
#define IDM_DATA_YBOUNDS  9025
#define IDM_RUNAPP        9030
#define IDM_HELP_ABOUT    9999

IDM_MAINMENU MENU DISCARDABLE
BEGIN
    POPUP "&File"
    BEGIN
        MENUITEM "E&xit...", IDM_FILE_EXIT
    END

    POPUP "&Data"
    BEGIN
        MENUITEM "Ball &Radius...", IDM_DATA_RADIUS
        MENUITEM "Ball &Speed...", IDM_DATA_SPEED
        MENUITEM SEPARATOR
        MENUITEM "Time &Interval...", IDM_DATA_TTOT
        MENUITEM "Time Ste&p...", IDM_DATA_TSTEP
        MENUITEM SEPARATOR
        MENUITEM "&X bounds...", IDM_DATA_XBOUNDS
        MENUITEM "&Y bounds...", IDM_DATA_YBOUNDS
    END

    POPUP "&Run Application"
    BEGIN
        MENUITEM "R&un", IDM_RUNAPP
    END

    POPUP "&Help"
    BEGIN
        MENUITEM "&About...", IDM_HELP_ABOUT
    END
END

#include <windows.h>

#define IDC_STATIC −1

#define IDD_DATA_RADIUS  100
#define IDC_RADIUS       101

IDD_DATA_RADIUS DIALOG DISCARDABLE  0, 0, 284, 77
STYLE DS_MODALFRAME | WS_POPUP | WS_CAPTION | WS_SYSMENU
CAPTION "Ball Radius"
FONT 8, "MS Sans Serif"
BEGIN
    DEFPUSHBUTTON   "OK", IDOK, 227, 7, 50, 14
    PUSHBUTTON      "Cancel", IDCANCEL, 227, 24, 50, 14
    CTEXT           "This program will shows a ball bouncing in a rectangle.",
                    IDC_STATIC, 7, 7, 153, 18
    GROUPBOX        "Ball &Radius", IDC_STATIC, 13, 30, 186, 34
    EDITTEXT        IDC_RADIUS, 35, 43, 60, 14, ES_AUTOHSCROLL
    LTEXT           " cm", IDC_STATIC, 97, 45, 20, 8
END
```

```
#define IDD_DATA_SPEED  150
#define IDC_SPEED       151

IDD_DATA_SPEED DIALOG DISCARDABLE  0, 0, 284, 77
STYLE DS_MODALFRAME | WS_POPUP | WS_CAPTION | WS_SYSMENU
CAPTION "Ball Radius"
FONT 8, "MS Sans Serif"
BEGIN
    DEFPUSHBUTTON   "OK", IDOK, 227, 7, 50, 14
    PUSHBUTTON      "Cancel", IDCANCEL, 227, 24, 50, 14
    CTEXT           "This program will shows a ball bouncing in a rectangle.",
                    IDC_STATIC, 7, 7, 153, 18
    GROUPBOX        "Ball &Speed", IDC_STATIC, 13, 30, 186, 34
    EDITTEXT        IDC_SPEED, 35, 43, 60, 14, ES_AUTOHSCROLL
    LTEXT           " cm/s", IDC_STATIC, 97, 45, 20, 8
END

#define IDD_DATA_TTOT  200
#define IDC_TMIN       201
#define IDC_TMAX       202

IDD_DATA_TTOT DIALOG DISCARDABLE  0, 0, 284, 97
STYLE DS_MODALFRAME | WS_POPUP | WS_CAPTION | WS_SYSMENU
CAPTION "Time Interval"
FONT 8, "MS Sans Serif"
BEGIN
    DEFPUSHBUTTON   "OK", IDOK, 227, 7, 50, 14
    PUSHBUTTON      "Cancel", IDCANCEL, 227, 24, 50, 14
    CTEXT           "This program will shows a ball bouncing in a rectangle.",
                    IDC_STATIC, 7, 7, 153, 18
    GROUPBOX        "Time &Interval", IDC_STATIC, 13, 30, 186, 54
    LTEXT           "TM&IN : ", IDC_STATIC, 35, 45, 30, 8
    EDITTEXT        IDC_TMIN, 85, 43, 60, 14, ES_AUTOHSCROLL
    LTEXT           " s", IDC_STATIC, 147, 45, 20, 8
    LTEXT           "TM&AX : ", IDC_STATIC, 35, 65, 30, 8
    EDITTEXT        IDC_TMAX, 85, 63, 60, 14, ES_AUTOHSCROLL
    LTEXT           " s", IDC_STATIC, 147, 65, 20, 8
END

#define IDD_DATA_TSTEP   300
#define IDC_TSTEP        301

IDD_DATA_TSTEP DIALOG DISCARDABLE  0, 0, 284, 77
STYLE DS_MODALFRAME | WS_POPUP | WS_CAPTION | WS_SYSMENU
CAPTION "Time Step"
FONT 8, "MS Sans Serif"
BEGIN
    DEFPUSHBUTTON   "OK", IDOK, 227, 7, 50, 14
    PUSHBUTTON      "Cancel", IDCANCEL, 227, 24, 50, 14
    CTEXT           "This program will shows a ball bouncing in a rectangle.",
                    IDC_STATIC, 7, 7, 153, 18
    GROUPBOX        "Time &Step", IDC_STATIC, 13, 30, 186, 34
    EDITTEXT        IDC_TSTEP, 35, 43, 60, 14, ES_AUTOHSCROLL
    LTEXT           " s", IDC_STATIC, 97, 45, 20, 8
END

#define IDD_DATA_XBOUNDS  400
#define IDC_XMIN          401
#define IDC_XMAX          402

IDD_DATA_XBOUNDS DIALOG DISCARDABLE  0, 0, 284, 97
STYLE DS_MODALFRAME | WS_POPUP | WS_CAPTION | WS_SYSMENU
CAPTION "X bounds"
FONT 8, "MS Sans Serif"
BEGIN
    DEFPUSHBUTTON   "OK", IDOK, 227, 7, 50, 14
    PUSHBUTTON      "Cancel", IDCANCEL, 227, 24, 50, 14
    CTEXT           "This program will shows a ball bouncing in a rectangle.",
                    IDC_STATIC, 7, 7, 153, 18
    GROUPBOX        "&X Bounds", IDC_STATIC, 13, 30, 186, 54
    LTEXT           "XM&IN : ", IDC_STATIC, 35, 45, 30, 8
    EDITTEXT        IDC_XMIN, 85, 43, 60, 14, ES_AUTOHSCROLL
    LTEXT           " cm", IDC_STATIC, 147, 45, 20, 8
    LTEXT           "XM&AX : ", IDC_STATIC, 35, 65, 30, 8
    EDITTEXT        IDC_XMAX, 85, 63, 60, 14, ES_AUTOHSCROLL
    LTEXT           " cm", IDC_STATIC, 147, 65, 20, 8
END
```

```
#define IDD_DATA_YBOUNDS  500
#define IDC_YMIN          501
#define IDC_YMAX          502

IDD_DATA_YBOUNDS DIALOG DISCARDABLE  0, 0, 284, 97
STYLE DS_MODALFRAME | WS_POPUP | WS_CAPTION | WS_SYSMENU
CAPTION "Y bounds"
FONT 8, "MS Sans Serif"
BEGIN
    DEFPUSHBUTTON   "OK", IDOK, 227, 7, 50, 14
    PUSHBUTTON      "Cancel", IDCANCEL, 227, 24, 50, 14
    CTEXT           "This program will shows a ball bouncing in a rectangle.",
                    IDC_STATIC, 7, 7, 153, 18
    GROUPBOX        "&Y Bounds", IDC_STATIC, 13, 30, 186, 54
    LTEXT           "YM&IN : ", IDC_STATIC, 35, 45, 30, 8
    EDITTEXT        IDC_YMIN, 85, 43, 60, 14, ES_AUTOHSCROLL
    LTEXT           " cm", IDC_STATIC, 147, 45, 20, 8
    LTEXT           "YM&AX : ", IDC_STATIC, 35, 65, 30, 8
    EDITTEXT        IDC_YMAX, 85, 63, 60, 14, ES_AUTOHSCROLL
    LTEXT           " cm", IDC_STATIC, 147, 65, 20, 8
END


#define IDD_ABOUT  999

IDD_ABOUT DIALOG DISCARDABLE  0, 0, 239, 66
STYLE DS_MODALFRAME | WS_POPUP | WS_CAPTION | WS_SYSMENU
CAPTION "About Box"
FONT 8, "MS Sans Serif"
BEGIN
    DEFPUSHBUTTON   "OK", IDOK, 174, 18, 50, 14
    PUSHBUTTON      "Cancel", IDCANCEL, 174, 35, 50, 14
    GROUPBOX        "About this program...", IDC_STATIC, 7, 7, 225, 52
    CTEXT           "A Double Buffering Method Demo\n\nby Angelo Graziosi",
                    IDC_STATIC, 16, 18, 144, 33
END
```

```
!
! (Partial) Fortran Interface to the Windows API Library
! by Angelo Graziosi  (firstname.lastnameATalice.it)
! Copyright Angelo Graziosi
!
! It is distributed in the hope that it will be useful,
! but WITHOUT ANY WARRANTY; without even the implied warranty of
! MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
!
! HOW TO BUILD (MSYS2/MINGW32/MINGW64 shell)
!
!   cd ~/programming/win32−fortran/bounce_plus
!
!   rm −rf {*.mod,*.res,~/programming/modules/*} && \
!   windres bounce_plus.rc −O coff −o bounce_plus.res && \
!   gfortran −O3 −Wall −mwindows −J ~/programming/modules \
!     ~/programming/basic−modules/basic_mods.f90 \
!     ../{win32.f90,win32boxes.f90,win32app.f90} rseed_rand.f90 \
!     bounce_plus.f90 bounce_plus.res −o bounce_plus.out && \
!     rm −rf {*.mod,*.res,~/programming/modules/*}
!
! In MINGW32/MINGW64, add '−static' and:
!
!   bounce_plus.out ==> bounce_plus−mingw32/mingw64
!
!
! Remember that:
!
!   int(0,UINT_T)    −−>  0_UINT_T
!   int(0,WPARAM_T)  −−>  0_WPARAM_T
!   int(0,LPARAM_T)  −−>  0_LPARAM_T
!

module the_app
  use kind_consts, only: DP
  use AboutBox_class
  use XBox_class
  use XYBox_class
  use RadioBox_class
  use win32, only: BLACK_COLOR, BLACKNESS, COLORREF_T, CYAN_COLOR, BOOL_T, &
       DWORD_T, FALSE_T, HBITMAP_T, HBRUSH_T, HDC_T, HINSTANCE_T, &
       HOLLOW_BRUSH, HPEN_T, HWND_T, IDYES, INT_T, LPARAM_T, LRESULT_T, &
       MAX_FMT, MAX_LEN, NL, NUL, NULL_T, PS_SOLID, TRUE_T, UINT_T, &
       WHITE_COLOR, WM_CLOSE, WM_COMMAND, WM_CREATE, WM_DESTROY, WM_SIZE, &
       WORD_T, WPARAM_T, YELLOW_COLOR, &
       ask_confirmation, CreateCompatibleDC, CreateHatchBrush, CreatePen, &
       DefWindowProc, DeleteDC, DeleteObject, DestroyWindow, error_msg, &
       GetDC, GetStockObject, lo_word, MessageBeep, PostMessage, &
       PostQuitMessage, ReleaseDC, &
       RGB, SelectObject, SetBkColor, SetTextColor, TextOut
  use win32app, only: win32app_BitBlt, win32app_clearDC, &
       win32app_CreateCompatibleBitmap, win32app_ellipse, win32app_setup, &
       win32app_xbounds, win32app_ybounds, &
       win32app_xmin, win32app_xmax, win32app_ymin, win32app_ymax
  implicit none
  private

  integer(WORD_T), parameter, public :: IDI_BOUNCE_PLUS = 1
  integer(WORD_T), parameter, public :: IDM_MAINMENU    = 9000

  integer(WORD_T), parameter :: IDM_FILE_EXIT      = 9010
  integer(WORD_T), parameter :: IDM_DATA_NBALLS    = 9020
  integer(WORD_T), parameter :: IDM_DATA_DENSITY   = 9021
  integer(WORD_T), parameter :: IDM_DATA_STIFFNES  = 9022
  integer(WORD_T), parameter :: IDM_DATA_MBOUNDS   = 9023
  integer(WORD_T), parameter :: IDM_DATA_TTOT      = 9024
  integer(WORD_T), parameter :: IDM_DATA_TSTEP     = 9025
  integer(WORD_T), parameter :: IDM_DATA_XBOUNDS   = 9026
  integer(WORD_T), parameter :: IDM_DATA_YBOUNDS   = 9027
  integer(WORD_T), parameter :: IDM_OPTIONS_TCOLOR = 9030
  integer(WORD_T), parameter :: IDM_RUNAPP         = 9040
  integer(WORD_T), parameter :: IDM_HELP_ABOUT     = 9999

  !integer(WORD_T), parameter :: IDC_STATIC = −1

  integer(WORD_T), parameter :: IDD_DATA_NBALLS = 100
  integer(INT_T), parameter :: IDC_NBALLS       = 101
```

```
  integer(WORD_T), parameter :: IDD_DATA_DENSITY = 200
  integer(INT_T), parameter :: IDC_DENSITY       = 201

  integer(WORD_T), parameter :: IDD_DATA_STIFFNES = 300
  integer(INT_T), parameter :: IDC_STIFFNES       = 301

  integer(WORD_T), parameter :: IDD_DATA_MBOUNDS = 400
  integer(INT_T), parameter :: IDC_MMIN          = 401
  integer(INT_T), parameter :: IDC_MMAX          = 402

  integer(WORD_T), parameter :: IDD_DATA_TTOT = 500
  integer(INT_T), parameter :: IDC_TMIN       = 501
  integer(INT_T), parameter :: IDC_TMAX       = 502

  integer(WORD_T), parameter :: IDD_DATA_TSTEP = 600
  integer(INT_T), parameter :: IDC_TSTEP       = 601

  integer(WORD_T), parameter :: IDD_DATA_XBOUNDS = 700
  integer(INT_T), parameter :: IDC_XMIN          = 701
  integer(INT_T), parameter :: IDC_XMAX          = 702

  integer(WORD_T), parameter :: IDD_DATA_YBOUNDS = 800
  integer(INT_T), parameter :: IDC_YMIN          = 801
  integer(INT_T), parameter :: IDC_YMAX          = 802

  integer(WORD_T), parameter :: IDD_OPTIONS_TCOLOR = 900
  integer(INT_T), parameter :: IDC_CYAN            = 901
  integer(INT_T), parameter :: IDC_WHITE           = 902
  integer(INT_T), parameter :: IDC_YELLOW          = 903

  integer(WORD_T), parameter :: IDD_ABOUT = 999

  type ball_type
     integer(COLORREF_T) :: col = BLACK_COLOR
     real(DP) :: mass = 0.0_DP, &
          density = 0.0_DP, &
          radius = 0.0_DP
     real(DP), dimension(2) :: frc = 0.0_DP, &
          acc = 0.0_DP, &
          vel = 0.0_DP, &
          pos = 0.0_DP
  end type ball_type

  ! COMMON data
  integer(HBITMAP_T) :: hBitmap = NULL_T
  logical :: run_flag = .true.
  real(DP) :: box_xmin, box_xmax, box_ymin, box_ymax

  ! Application data, strictly speaking...
  integer :: nballs = 12
  real(DP) :: density = 0.01_DP, stiffnes = 5E5_DP
  real(DP) :: m0 = 400.0_DP, m1 = 8000.0_DP
  type(ball_type), allocatable :: ball(:)

  real(DP) :: t = 0.0_DP, t0 = 0.0_DP, t1 = 900.0_DP, &
       tstep = 1.0_DP/512  ! 1.953125E-03 = 0.000000001_2

  integer :: tcolor = 2  ! WHITE

  public :: paint_screen, WndProc

contains

  subroutine balls_on()
    integer :: ierr
    allocate(ball(nballs),stat=ierr)
    if (ierr /= 0) then
       write(*,*) '*** FATAL ERROR ***'
       write(*,*) 'BALL: Allocation request denied'
       stop
    end if
  end subroutine balls_on

  subroutine balls_off()
    integer :: ierr
    if (allocated(ball)) deallocate(ball,stat=ierr)
    if (ierr /= 0) then
       write(*,*) '*** FATAL ERROR ***'
```

```
      write(*,*) 'BALL: Deallocation request denied'
      stop
    end if
  end subroutine balls_off

  subroutine setup_balls()
    use math_consts, only: PI
    real(DP), parameter :: Z3 = 1.0_DP/3, Z43PI = 4*Z3*PI
    real(DP) :: u(9)
    integer  :: i

    ! Time initialization
    t = t0

    ! Set startup conditions of elastic balls
    do i = 1, nballs
       call random_number(u)
       ball(i)%col = RGB(int(64+u(1)*192),int(64+u(2)*192),int(64+u(3)*192))
       ball(i)%mass = m0+(i−1)*(m1−m0)/(nballs−1)
       ball(i)%density = density
       ball(i)%radius = ((ball(i)%mass/ball(i)%density)/(Z43PI))**Z3
       ball(i)%pos = [ (1.0_DP−u(4))*(box_xmin+ball(i)%radius) &
            +u(4)*(box_xmax−ball(i)%radius), &
            (1.0_DP−u(5))*(box_ymin+ball(i)%radius) &
            +u(5)*(box_ymax−ball(i)%radius) ]
       ball(i)%vel = 200*[ u(6)−u(7), u(8)−u(9) ]
    end do
  end subroutine setup_balls

  subroutine draw_time(hdc,t)
    integer(HDC_T), intent(in) :: hdc
    real(DP), intent(in) :: t
    integer(COLORREF_T), parameter :: TXT_COLOR(3) = &
         [ CYAN_COLOR, WHITE_COLOR, YELLOW_COLOR ]
    ! We use SAVE just to save something at each call
    ! (draw_time() is called intensively, at each iteration)
    integer(COLORREF_T), save :: old_bk_color, old_text_color
    character(len=MAX_LEN), save :: buffer = ''
    integer, save :: dummy

    old_bk_color = SetBkColor(hdc,BLACK_COLOR)
    old_text_color = SetTextColor(hdc,TXT_COLOR(tcolor))

    buffer = ''
    write(buffer,*) 'Time : ',t
    buffer = trim(adjustl(buffer))//'   '//NUL
    dummy = index(buffer,NUL)
    !dummy = TextOut(hdc,xs(x_min),ys(y_max),buffer(1:dummy),dummy−1)
    dummy = TextOut(hdc,0,0,buffer(1:dummy),dummy−1)

    ! Restore previous text colors...
    dummy = SetBkColor(hdc,old_bk_color)
    dummy = SetTextColor(hdc,old_text_color)
  end subroutine draw_time

  subroutine draw_ball(hdc,p,r,col)
    integer(HDC_T), intent(in) :: hdc
    real(DP), intent(in) :: p(:), r
    integer(COLORREF_T), intent(in) :: col
    ! We use SAVE just to save something at each call
    ! (draw_ball() is called intensively, at each iteration)
    integer(HPEN_T), save :: hPen
    integer, save :: dummy

    ! Set the fill style
    dummy = int(SelectObject(hdc,GetStockObject(HOLLOW_BRUSH)),INT_T)

    hPen = CreatePen(PS_SOLID,1,col)
    dummy = int(SelectObject(hdc,hPen),INT_T)

    dummy = win32app_ellipse(hdc,p(1)−r,p(2)+r,p(1)+r,p(2)−r)

    dummy = win32app_ellipse(hdc,p(1)−(r−0.5_DP),p(2)+(r−0.5_DP), &
         p(1)+(r−0.5_DP),p(2)−(r−0.5_DP))

    dummy = win32app_ellipse(hdc,p(1)−(r−1.0_DP),p(2)+(r−1.0_DP), &
         p(1)+(r−1.0_DP),p(2)−(r−1.0_DP))
```

```
  dummy = DeleteObject(hPen)
end subroutine draw_ball

subroutine painting_setup(hWnd)
  integer(HWND_T), intent(in) :: hWnd
  logical, save :: first = .true.
  integer(HDC_T) :: hdc, hdcMem
  integer :: dummy, i

  if (first) then
     call setup_balls()
     first = .false.
  end if

  if (hBitmap /= NULL_T) then
     dummy = DeleteObject(hBitmap)
  end if

  hdc = GetDC(hWnd)
  hdcMem = CreateCompatibleDC(hdc)
  hBitmap = win32app_CreateCompatibleBitmap(hdc)
  dummy = ReleaseDC(hWnd,hdc)

  dummy = int(SelectObject(hdcMem,hBitmap),INT_T)

  ! Clear the off−screen DC (hdcMem) for the next drawing
  dummy = win32app_clearDC(hdcMem,BLACKNESS)

  ! Draw (on the off−screen DC) time and elastic balls at time t
  call draw_time(hdcMem,t)
  do i = 1, nballs
     call draw_ball(hdcMem,ball(i)%pos,ball(i)%radius,ball(i)%col)
  end do

  dummy = DeleteDC(hdcMem)
end subroutine painting_setup

subroutine set_nballs(hWnd)
  integer(HWND_T), intent(in) :: hWnd
  character(len=MAX_FMT), parameter :: FMT = '(f12.0)'
  type(XBox) :: xb

  call new_box(xb,hWnd,IDD_DATA_NBALLS,IDC_NBALLS,FMT,real(nballs,DP))

  if (run(xb) > 0) then

     ! Destroying the current balls...
     call balls_off()

     ! We need nballs > 0, at least...
     nballs = int(abs(get(xb)))

     if (nballs < 2) then
        call error_msg('NBalls < 2 !!!'//NL &
             //'You need at least 2 balls...   '//NUL)
        nballs = 2
     end if

     ! Creating the new balls...
     call balls_on()

     ! If you prefer to see something, uncomment the following...
     !call setup_balls()
  end if
end subroutine set_nballs

subroutine set_density(hWnd)
  integer(HWND_T), intent(in) :: hWnd
  character(len=MAX_FMT), parameter :: FMT = '(1pg12.5)'
  type(XBox) :: xb

  call new_box(xb,hWnd,IDD_DATA_DENSITY,IDC_DENSITY,FMT,density)

  if (run(xb) > 0) then
     density = get(xb)

     if (density < 0) then
        call error_msg('Density < 0 !!!'//NL &
```

```
            //'Taking its absolute value...   '//NUL)
         density = abs(density)
      end if
   end if
end subroutine set_density

subroutine set_stiffnes(hWnd)
   integer(HWND_T), intent(in) :: hWnd
   character(len=MAX_FMT), parameter :: FMT = '(1pg12.5)'
   type(XBox) :: xb

   call new_box(xb,hWnd,IDD_DATA_STIFFNES,IDC_STIFFNES,FMT,stiffnes)

   if (run(xb) > 0) then
      stiffnes = get(xb)

      if (stiffnes < 0) then
         call error_msg('Stiffnes < 0 !!!'//NL &
              //'Taking its absolute value...   '//NUL)
         stiffnes = abs(stiffnes)
      end if
   end if
end subroutine set_stiffnes

subroutine set_massbounds(hWnd)
   integer(HWND_T), intent(in) :: hWnd
   character(len=MAX_FMT), parameter :: FMT = '(1pg12.5)'
   type(XYBox) :: xyb

   call new_box(xyb,hWnd,IDD_DATA_MBOUNDS,IDC_MMIN,IDC_MMAX,FMT,m0,m1)

   if (run(xyb) > 0) then
      m0  = min(abs(get_x(xyb)),abs(get_y(xyb)))
      m1  = max(abs(get_x(xyb)),abs(get_y(xyb)))
   end if
end subroutine set_massbounds

subroutine set_timebounds(hWnd)
   integer(HWND_T), intent(in) :: hWnd
   character(len=MAX_FMT), parameter :: FMT = '(1pg12.5)'
   type(XYBox) :: xyb

   call new_box(xyb,hWnd,IDD_DATA_TTOT,IDC_TMIN,IDC_TMAX,FMT,t0,t1)

   if (run(xyb) > 0) then
      t0  = min(get_x(xyb),get_y(xyb))
      t1  = max(get_x(xyb),get_y(xyb))
   end if
end subroutine set_timebounds

subroutine set_tstep(hWnd)
   integer(HWND_T), intent(in) :: hWnd
   character(len=MAX_FMT), parameter :: FMT = '(1pg12.5)'
   type(XBox) :: xb

   call new_box(xb,hWnd,IDD_DATA_TSTEP,IDC_TSTEP,FMT,tstep)

   if (run(xb) > 0) then
      tstep  = get(xb)

      if (tstep < 0) then
         call error_msg('TStep < 0 !!!'//NL &
              //'Taking its absolute value...   '//NUL)
         tstep = abs(tstep)
      end if
   end if
end subroutine set_tstep

subroutine set_tcolor(hWnd)
   integer(HWND_T), intent(in) :: hWnd
   integer, parameter :: NUM_BUTTONS = 3
   character(len=*), parameter :: BUTTON_NAMES(NUM_BUTTONS) = [ &
        '&Cyan  ', &
        '&White ', &
        '&Yellow' ]
   type(RadioBox) :: rb

   call new_box(rb,hWnd,IDD_OPTIONS_TCOLOR,IDC_CYAN,BUTTON_NAMES, &
```

```
        NUM_BUTTONS,tcolor)

   if (run(rb) > 0) tcolor = get_current_button(rb)
   !print *, 'TCOLOR = ',tcolor
 end subroutine set_tcolor

 subroutine help_dlg(hWnd)
   integer(HWND_T), intent(in) :: hWnd
   type(AboutBox) :: ab
   integer :: dummy
   call new_box(ab,hWnd,IDD_ABOUT)
   dummy = run(ab)
 end subroutine help_dlg

 function process_command(hWnd,wParam)
   integer(BOOL_T) :: process_command
   integer(HWND_T), intent(in) :: hWnd
   integer(WPARAM_T), intent(in) :: wParam
   integer :: dummy

   run_flag = .false.

   select case(lo_word(int(wParam,DWORD_T)))
   case(IDM_FILE_EXIT)
      dummy = MessageBeep(64)
      if (ask_confirmation(hWnd,'Sure you want to exit?   '//NUL, &
           'Exit?'//NUL) == IDYES) then
         dummy = PostMessage(hWnd,WM_CLOSE,0_WPARAM_T,0_LPARAM_T)
      end if
      process_command = TRUE_T
      return

   case(IDM_DATA_NBALLS)
      call set_nballs(hWnd)
      process_command = TRUE_T
      return

   case(IDM_DATA_DENSITY)
      call set_density(hWnd)
      process_command = TRUE_T
      return

   case(IDM_DATA_STIFFNES)
      call set_stiffnes(hWnd)
      process_command = TRUE_T
      return

   case(IDM_DATA_MBOUNDS)
      call set_massbounds(hWnd)
      process_command = TRUE_T
      return

   case(IDM_DATA_TTOT)
      call set_timebounds(hWnd)
      process_command = TRUE_T
      return

   case(IDM_DATA_TSTEP)
      call set_tstep(hWnd)
      process_command = TRUE_T
      return

   case(IDM_DATA_XBOUNDS)
      call win32app_xbounds(hWnd,IDD_DATA_XBOUNDS,IDC_XMIN,IDC_XMAX)
      process_command = TRUE_T
      return

   case(IDM_DATA_YBOUNDS)
      call win32app_ybounds(hWnd,IDD_DATA_YBOUNDS,IDC_YMIN,IDC_YMAX)
      process_command = TRUE_T
      return

   case(IDM_OPTIONS_TCOLOR)
      call set_tcolor(hWnd)
      process_command = TRUE_T
      return

   case(IDM_RUNAPP)
```

```
      run_flag = .true.
      call setup_balls()
      process_command = TRUE_T
      return

   case(IDM_HELP_ABOUT)
      call help_dlg(hWnd)
      process_command = TRUE_T
      return

   case default
      process_command = FALSE_T
      return
   end select
end function process_command

function WndProc(hWnd,iMsg,wParam,lParam) bind(C)
   !GCC$ ATTRIBUTES STDCALL :: WndProc
   integer(LRESULT_T) :: WndProc
   integer(HWND_T), value :: hWnd
   integer(UINT_T), value :: iMsg
   integer(WPARAM_T), value :: wParam
   integer(LPARAM_T), value :: lParam

   logical, save :: first = .true.
   integer :: dummy

   select case(iMsg)
   case(WM_CREATE)
      ! Creating the balls...
      call balls_on()

      WndProc = 0
      return

   case(WM_SIZE)
      if (first) then
         call win32app_setup(lParam,-600.0_DP,600.0_DP)
         first = .false.
      else
         call win32app_setup(lParam)
      end if

      ! Getting the box boundaries... each time, maybe, the mapping changed...
      box_xmin = win32app_xmin()
      box_xmax = win32app_xmax()
      box_ymin = win32app_ymin()
      box_ymax = win32app_ymax()

      ! Now that the mapping has been defined, we can initialize the painting
      call painting_setup(hWnd)

      WndProc = 0
      return

   case(WM_COMMAND)
      if (process_command(hWnd,wParam) == TRUE_T) then
         WndProc = 0
         return
      end if
      ! ...else it continues with DefWindowProc

   case(WM_CLOSE)
      dummy = DestroyWindow(hWnd)
      WndProc = 0
      return

   case(WM_DESTROY)
      if (hBitmap /= NULL_T) then
         dummy = DeleteObject(hBitmap)
      end if

      ! Destroying the balls...
      call balls_off()

      call PostQuitMessage(0)
      ! Commenting out the next two statements, it continues
      ! with DefWindowProc()
```

```
      WndProc = 0
      return
   end select

   WndProc = DefWindowProc(hWnd,iMsg,wParam,lParam)
end function WndProc

subroutine update_ball_position()
   ! We use SAVE just to save something at each call
   ! (update_ball_position() is called intensively, at each iteration)
   real(DP), save :: force(2), ball_distance, dist_min, dst(2)
   integer, save :: i, j

   ! Test all elastic balls against each other.
   ! Calculate forces if they touch.
   do i = 1, nballs−1
      do j = i+1, nballs
         ! Distance between elastic balls (Pythagoras' theorem)
         dst = ball(j)%pos−ball(i)%pos
         ball_distance = norm2(dst)
         dist_min = ball(i)%radius+ball(j)%radius
         if (ball_distance < dist_min) then

            ! Cosine and sine to the angle between ball i and j
            ! (trigonometry): here 'force' is a unit vector!
            force = dst/ball_distance

            ! Spring force (Hooke's law of elasticity)
            ! Here 'force' is the total force of 'i' on 'j' :
            ! (All capital letters are vectors)
            !
            !   F(i −> j) = −k * S = −k*(Bd−Dm) = −k*(|Bd|−|Dm|)*U
            !   U = Bd/|Bd|
            force = −stiffnes*(ball_distance−dist_min)*force

            ! F(i) = F(i)+F(j,i) = F(i)−F(i,j), F(j) = F(j)+F(i,j)
            ! being F(i,j) the force of 'i' on 'j'
            ball(i)%frc = ball(i)%frc−force
            ball(j)%frc = ball(j)%frc+force
         end if
      end do
   end do

   ! Update acceleration, velocity, and position of elastic balls
   ! (using the Euler−Cromer 1st order integration algorithm)
   do i = 1, nballs
      ! Accelerate balls (acceleration = force / mass)
      ball(i)%acc = ball(i)%frc/ball(i)%mass

      ! Reset force vector
      ball(i)%frc = 0.0_DP

      ! Update velocity
      ! delta velocity = acceleration * delta time
      ! new velocity = old velocity + delta velocity
      ball(i)%vel = ball(i)%vel+ball(i)%acc*tstep

      ! Update position
      ! delta position = velocity * delta time
      ! new position = old position + delta position
      ball(i)%pos = ball(i)%pos+ball(i)%vel*tstep
   end do

   ! Keep elastic balls within screen boundaries
   do i = 1, nballs
      ! Right
      if (ball(i)%pos(1) > box_xmax−ball(i)%radius) then
         ball(i)%vel(1) = −ball(i)%vel(1)
         ball(i)%pos(1) = box_xmax−ball(i)%radius
      end if

      ! Left
      if (ball(i)%pos(1) < box_xmin+ball(i)%radius) then
         ball(i)%vel(1) = −ball(i)%vel(1)
         ball(i)%pos(1) = box_xmin+ball(i)%radius
      end if

      ! Top
```

```
      if (ball(i)%pos(2) > box_ymax−ball(i)%radius) then
         ball(i)%vel(2) = −ball(i)%vel(2)
         ball(i)%pos(2) = box_ymax−ball(i)%radius
      end if

      ! Bottom
      if (ball(i)%pos(2) < box_ymin+ball(i)%radius) then
         ball(i)%vel(2) = −ball(i)%vel(2)
         ball(i)%pos(2) = box_ymin+ball(i)%radius
      end if
    end do
  end subroutine update_ball_position

  subroutine paint_screen(hWnd)
    integer(HWND_T), intent(in) :: hWnd
    ! We use SAVE just to save something at each call
    ! (paint_screen() is called intensively, at each iteration)
    integer(HDC_T), save :: hdc, hdcMem
    integer, save :: dummy, i

    if (hBitmap /= NULL_T) then
       hdc = GetDC(hWnd)
       hdcMem = CreateCompatibleDC(hdc)
       dummy = int(SelectObject(hdcMem,hBitmap),INT_T)

       ! Transfer the off−screen DC to the screen
       dummy = win32app_BitBlt(hdc,hdcMem)
       dummy = ReleaseDC(hWnd,hdc)

       if (t < t1 .and. run_flag) then
          t = t+tstep
          call update_ball_position()
       end if

       ! Clear the off−screen DC (hdcMem) for the next drawing
       dummy = win32app_clearDC(hdcMem,BLACKNESS)

       ! Draw (on the off−screen DC) time and elastic balls at time t
       call draw_time(hdcMem,t)
       do i = 1, nballs
          call draw_ball(hdcMem,ball(i)%pos,ball(i)%radius,ball(i)%col)
       end do

       dummy = DeleteDC(hdcMem)
    end if
  end subroutine paint_screen
end module the_app

function WinMain(hInstance,hPrevInstance,lpCmdLine,nCmdShow) &
     bind(C, name='WinMain')
  use rseed_rand
  use, intrinsic :: iso_c_binding, only: C_PTR, C_CHAR, c_sizeof, c_funloc, &
       C_FUNPTR, c_loc
  use win32, only: BLACK_BRUSH, CS_HREDRAW, CS_VREDRAW, CW_USEDEFAULT, &
       DWORD_T, HINSTANCE_T, HWND_T, INT_T, NUL, NULL_PTR_T, NULL_T, &
       PM_REMOVE, WM_QUIT, WS_OVERLAPPEDWINDOW, UINT_T, &
       MSG_T, WNDCLASSEX_T, &
       arrow_cursor, CreateWindowEx, DispatchMessage, error_msg, ExitProcess, &
       GetStockObject, LoadCursor, LoadIcon, make_int_resource, &
       make_int_resource_C_PTR, PeekMessage, RegisterClassEx, ShowWindow, &
       TranslateMessage, UpdateWindow
  use the_app, only: IDI_BOUNCE_PLUS, IDM_MAINMENU, &
       paint_screen, WndProc
  use rseed_rand
  implicit none
  !GCC$ ATTRIBUTES STDCALL :: WinMain
  integer(INT_T) :: WinMain
  integer(HINSTANCE_T), value :: hInstance
  integer(HINSTANCE_T), value :: hPrevInstance
  type(C_PTR), value :: lpCmdLine ! LPSTR
  integer(INT_T), value :: nCmdShow

  character(kind=C_CHAR, len=128), target :: app_name = &
       'Bounce_Plus'//NUL
  character(kind=C_CHAR, len=*), parameter :: WINDOW_CAPTION = &
       'Bouncing Balls'//NUL
  type(WNDCLASSEX_T) :: WndClass
  integer(HWND_T) :: hWnd
```

```
  type(MSG_T) :: msg
  integer :: dummy

  ! To avoid some annoying warnings...
  integer(HINSTANCE_T) :: not_used_hPrevInstance
  type(C_PTR) :: not_used_lpCmdLine
  not_used_hPrevInstance = hPrevInstance
  not_used_lpCmdLine = lpCmdLine

  call rseed()

  WndClass%cbSize = int(c_sizeof(Wndclass),UINT_T)
  WndClass%style = ior(CS_HREDRAW,CS_VREDRAW)
  WndClass%lpfnWndProc = c_funloc(WndProc)
  WndClass%cbClsExtra = 0
  WndClass%cbWndExtra = 0
  WndClass%hInstance = hInstance
  WndClass%hIcon = LoadIcon(hInstance,make_int_resource(IDI_BOUNCE_PLUS))
  WndClass%hCursor = LoadCursor(NULL_T,arrow_cursor())
  WndClass%hbrBackground = GetStockObject(BLACK_BRUSH)
  WndClass%lpszMenuName = make_int_resource_C_PTR(IDM_MAINMENU)
  WndClass%lpszClassName = c_loc(app_name(1:1))
  WndClass%hIconSm = LoadIcon(hInstance,make_int_resource(IDI_BOUNCE_PLUS))

  if (RegisterClassEx(WndClass) == 0) then
     call error_msg('Window Registration Failure!   '//NUL)
     call ExitProcess(0_UINT_T)
     WinMain = 0
     !return
  end if

  hWnd = CreateWindowEx(0_DWORD_T, &
       app_name, &
       WINDOW_CAPTION, &
       WS_OVERLAPPEDWINDOW, &
       CW_USEDEFAULT,CW_USEDEFAULT,CW_USEDEFAULT,CW_USEDEFAULT, &
       NULL_T,NULL_T,hInstance,NULL_PTR_T)

  if (hWnd == NULL_T) then
     call error_msg('Window Creation Failure!   '//NUL)
     call ExitProcess(0_UINT_T)
     WinMain = 0
     !return
  end if

  dummy = ShowWindow(hWnd,nCmdShow)
  dummy = UpdateWindow(hWnd)

  ! See: Charles Petzold "Programming Windows", 5th ed., pag. 162
  ! 'Random Rectangles'
  do
     if (PeekMessage(msg,NULL_T,0,0,PM_REMOVE) /= 0) then
        if (msg%message == WM_QUIT) exit
        dummy = TranslateMessage(msg)
        dummy = int(DispatchMessage(msg),INT_T)
     else
        call paint_screen(hWnd)
     end if
  end do

  call ExitProcess(int(msg%wParam,UINT_T))
  WinMain = 0
end function WinMain
```

```
//
// (Partial) Fortran Interface to the Windows API Library
// by Angelo Graziosi  (firstname.lastnameATalice.it)
// Copyright Angelo Graziosi
//
// It is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
//
// RC file for "bounce_plus" app
//

#define IDI_BOUNCE_PLUS 1

IDI_BOUNCE_PLUS ICON DISCARDABLE "../common_icons/smiling_sun.ico"

#define IDM_MAINMENU       9000
#define IDM_FILE_EXIT      9010
#define IDM_DATA_NBALLS    9020
#define IDM_DATA_DENSITY   9021
#define IDM_DATA_STIFFNES  9022
#define IDM_DATA_MBOUNDS   9023
#define IDM_DATA_TTOT      9024
#define IDM_DATA_TSTEP     9025
#define IDM_DATA_XBOUNDS   9026
#define IDM_DATA_YBOUNDS   9027
#define IDM_OPTIONS_TCOLOR 9030
#define IDM_RUNAPP         9040
#define IDM_HELP_ABOUT     9999

IDM_MAINMENU MENU DISCARDABLE
BEGIN
    POPUP "&File"
    BEGIN
        MENUITEM "E&xit...", IDM_FILE_EXIT
    END

    POPUP "&Data"
    BEGIN
        MENUITEM "&Number Of Balls...", IDM_DATA_NBALLS
        MENUITEM SEPARATOR
        MENUITEM "Ball &Density...", IDM_DATA_DENSITY
        MENUITEM "Spring Sti&ffnes...", IDM_DATA_STIFFNES
        MENUITEM "&Mass bounds...", IDM_DATA_MBOUNDS
        MENUITEM SEPARATOR
        MENUITEM "Time &Interval...", IDM_DATA_TTOT
        MENUITEM "Time &Step...", IDM_DATA_TSTEP
        MENUITEM SEPARATOR
        MENUITEM "&X bounds...", IDM_DATA_XBOUNDS
        MENUITEM "&Y bounds...", IDM_DATA_YBOUNDS
    END

    POPUP "&Options"
    BEGIN
        MENUITEM "Time &Color...", IDM_OPTIONS_TCOLOR
    END

    POPUP "&Run Application"
    BEGIN
        MENUITEM "R&un", IDM_RUNAPP
    END

    POPUP "&Help"
    BEGIN
        MENUITEM "&About...", IDM_HELP_ABOUT
    END
END

#include <windows.h>

#define IDC_STATIC −1

#define IDD_DATA_NBALLS  100
#define IDC_NBALLS       101

IDD_DATA_NBALLS DIALOG DISCARDABLE  0, 0, 284, 77
STYLE DS_MODALFRAME | WS_POPUP | WS_CAPTION | WS_SYSMENU
CAPTION "Number Of Balls"
```

```
FONT 8, "MS Sans Serif"
BEGIN
    DEFPUSHBUTTON   "OK", IDOK, 227, 7, 50, 14
    PUSHBUTTON      "Cancel", IDCANCEL, 227, 24, 50, 14
    CTEXT           "This program will shows bouncing balls in a box.",
                    IDC_STATIC, 7, 7, 153, 18
    GROUPBOX        "&Number Of Balls", IDC_STATIC, 13, 30, 186, 34
    EDITTEXT        IDC_NBALLS, 35, 43, 60, 14, ES_AUTOHSCROLL
    LTEXT           " ;−) ", IDC_STATIC, 97, 45, 20, 8
END

#define IDD_DATA_DENSITY  200
#define IDC_DENSITY       201

IDD_DATA_DENSITY DIALOG DISCARDABLE  0, 0, 284, 77
STYLE DS_MODALFRAME | WS_POPUP | WS_CAPTION | WS_SYSMENU
CAPTION "Ball Density"
FONT 8, "MS Sans Serif"
BEGIN
    DEFPUSHBUTTON   "OK", IDOK, 227, 7, 50, 14
    PUSHBUTTON      "Cancel", IDCANCEL, 227, 24, 50, 14
    CTEXT           "This program will shows bouncing balls in a box.",
                    IDC_STATIC, 7, 7, 153, 18
    GROUPBOX        "Ball &Density", IDC_STATIC, 13, 30, 186, 34
    EDITTEXT        IDC_DENSITY, 35, 43, 60, 14, ES_AUTOHSCROLL
    LTEXT           " g/cm**3", IDC_STATIC, 97, 45, 60, 8
END

#define IDD_DATA_STIFFNES  300
#define IDC_STIFFNES       301

IDD_DATA_STIFFNES DIALOG DISCARDABLE  0, 0, 284, 77
STYLE DS_MODALFRAME | WS_POPUP | WS_CAPTION | WS_SYSMENU
CAPTION "Spring Stiffnes"
FONT 8, "MS Sans Serif"
BEGIN
    DEFPUSHBUTTON   "OK", IDOK, 227, 7, 50, 14
    PUSHBUTTON      "Cancel", IDCANCEL, 227, 24, 50, 14
    CTEXT           "This program will shows bouncing balls in a box.",
                    IDC_STATIC, 7, 7, 153, 18
    GROUPBOX        "Spring Sti&ffnes", IDC_STATIC, 13, 30, 186, 34
    EDITTEXT        IDC_STIFFNES, 35, 43, 60, 14, ES_AUTOHSCROLL
    LTEXT           " dyn/cm", IDC_STATIC, 97, 45, 60, 8
END

#define IDD_DATA_MBOUNDS  400
#define IDC_MMIN          401
#define IDC_MMAX          402

IDD_DATA_MBOUNDS DIALOG DISCARDABLE  0, 0, 284, 97
STYLE DS_MODALFRAME | WS_POPUP | WS_CAPTION | WS_SYSMENU
CAPTION "Mass bounds"
FONT 8, "MS Sans Serif"
BEGIN
    DEFPUSHBUTTON   "OK", IDOK, 227, 7, 50, 14
    PUSHBUTTON      "Cancel", IDCANCEL, 227, 24, 50, 14
    CTEXT           "This program will shows bouncing balls in a box.",
                    IDC_STATIC, 7, 7, 153, 18
    GROUPBOX        "&Mass bounds", IDC_STATIC, 13, 30, 186, 54
    LTEXT           "MM&IN : ", IDC_STATIC, 35, 45, 30, 8
    EDITTEXT        IDC_MMIN, 85, 43, 60, 14, ES_AUTOHSCROLL
    LTEXT           " g", IDC_STATIC, 147, 45, 20, 8
    LTEXT           "MM&AX : ", IDC_STATIC, 35, 65, 30, 8
    EDITTEXT        IDC_MMAX, 85, 63, 60, 14, ES_AUTOHSCROLL
    LTEXT           " g", IDC_STATIC, 147, 65, 20, 8
END

#define IDD_DATA_TTOT  500
#define IDC_TMIN       501
#define IDC_TMAX       502

IDD_DATA_TTOT DIALOG DISCARDABLE  0, 0, 284, 97
STYLE DS_MODALFRAME | WS_POPUP | WS_CAPTION | WS_SYSMENU
CAPTION "Time Interval"
FONT 8, "MS Sans Serif"
BEGIN
    DEFPUSHBUTTON   "OK", IDOK, 227, 7, 50, 14
    PUSHBUTTON      "Cancel", IDCANCEL, 227, 24, 50, 14
```

```
    CTEXT           "This program will shows a ball bouncing in a rectangle.",
                    IDC_STATIC, 7, 7, 153, 18
    GROUPBOX        "Time &Interval", IDC_STATIC, 13, 30, 186, 54
    LTEXT           "TM&IN : ", IDC_STATIC, 35, 45, 30, 8
    EDITTEXT        IDC_TMIN, 85, 43, 60, 14, ES_AUTOHSCROLL
    LTEXT           " s", IDC_STATIC, 147, 45, 20, 8
    LTEXT           "TM&AX : ", IDC_STATIC, 35, 65, 30, 8
    EDITTEXT        IDC_TMAX, 85, 63, 60, 14, ES_AUTOHSCROLL
    LTEXT           " s", IDC_STATIC, 147, 65, 20, 8
END

#define IDD_DATA_TSTEP   600
#define IDC_TSTEP        601

IDD_DATA_TSTEP DIALOG DISCARDABLE  0, 0, 284, 77
STYLE DS_MODALFRAME | WS_POPUP | WS_CAPTION | WS_SYSMENU
CAPTION "Time Step"
FONT 8, "MS Sans Serif"
BEGIN
    DEFPUSHBUTTON   "OK", IDOK, 227, 7, 50, 14
    PUSHBUTTON      "Cancel", IDCANCEL, 227, 24, 50, 14
    CTEXT           "This program will shows a ball bouncing in a rectangle.",
                    IDC_STATIC, 7, 7, 153, 18
    GROUPBOX        "Time &Step", IDC_STATIC, 13, 30, 186, 34
    EDITTEXT        IDC_TSTEP, 35, 43, 60, 14, ES_AUTOHSCROLL
    LTEXT           " s", IDC_STATIC, 97, 45, 20, 8
END

#define IDD_DATA_XBOUNDS  700
#define IDC_XMIN          701
#define IDC_XMAX          702

IDD_DATA_XBOUNDS DIALOG DISCARDABLE  0, 0, 284, 97
STYLE DS_MODALFRAME | WS_POPUP | WS_CAPTION | WS_SYSMENU
CAPTION "X bounds"
FONT 8, "MS Sans Serif"
BEGIN
    DEFPUSHBUTTON   "OK", IDOK, 227, 7, 50, 14
    PUSHBUTTON      "Cancel", IDCANCEL, 227, 24, 50, 14
    CTEXT           "This program will shows a ball bouncing in a rectangle.",
                    IDC_STATIC, 7, 7, 153, 18
    GROUPBOX        "&X Bounds", IDC_STATIC, 13, 30, 186, 54
    LTEXT           "XM&IN : ", IDC_STATIC, 35, 45, 30, 8
    EDITTEXT        IDC_XMIN, 85, 43, 60, 14, ES_AUTOHSCROLL
    LTEXT           " cm", IDC_STATIC, 147, 45, 20, 8
    LTEXT           "XM&AX : ", IDC_STATIC, 35, 65, 30, 8
    EDITTEXT        IDC_XMAX, 85, 63, 60, 14, ES_AUTOHSCROLL
    LTEXT           " cm", IDC_STATIC, 147, 65, 20, 8
END

#define IDD_DATA_YBOUNDS  800
#define IDC_YMIN          801
#define IDC_YMAX          802

IDD_DATA_YBOUNDS DIALOG DISCARDABLE  0, 0, 284, 97
STYLE DS_MODALFRAME | WS_POPUP | WS_CAPTION | WS_SYSMENU
CAPTION "Y bounds"
FONT 8, "MS Sans Serif"
BEGIN
    DEFPUSHBUTTON   "OK", IDOK, 227, 7, 50, 14
    PUSHBUTTON      "Cancel", IDCANCEL, 227, 24, 50, 14
    CTEXT           "This program will shows a ball bouncing in a rectangle.",
                    IDC_STATIC, 7, 7, 153, 18
    GROUPBOX        "&Y Bounds", IDC_STATIC, 13, 30, 186, 54
    LTEXT           "YM&IN : ", IDC_STATIC, 35, 45, 30, 8
    EDITTEXT        IDC_YMIN, 85, 43, 60, 14, ES_AUTOHSCROLL
    LTEXT           " cm", IDC_STATIC, 147, 45, 20, 8
    LTEXT           "YM&AX : ", IDC_STATIC, 35, 65, 30, 8
    EDITTEXT        IDC_YMAX, 85, 63, 60, 14, ES_AUTOHSCROLL
    LTEXT           " cm", IDC_STATIC, 147, 65, 20, 8
END

#define IDD_OPTIONS_TCOLOR  900
#define IDC_CYAN            901
#define IDC_WHITE           902
#define IDC_YELLOW          903

IDD_OPTIONS_TCOLOR DIALOG DISCARDABLE  0, 0, 284, 117
```

```
STYLE DS_MODALFRAME | WS_POPUP | WS_CAPTION | WS_SYSMENU
CAPTION "Time Color"
FONT 8, "MS Sans Serif"
BEGIN
    DEFPUSHBUTTON   "OK", IDOK, 227, 7, 50, 14
    PUSHBUTTON      "Cancel", IDCANCEL, 227, 24, 50, 14
    CTEXT           "This program will shows bouncing balls in a box.",
                    IDC_STATIC, 7, 7, 153, 18
    GROUPBOX        "Time &Color", IDC_STATIC, 13, 30, 186, 74
    RADIOBUTTON     "&Cyan", IDC_CYAN, 35, 45, 60, 8
    RADIOBUTTON     "&White", IDC_WHITE, 35, 65, 60, 8
    RADIOBUTTON     "&Yellow", IDC_YELLOW, 35, 85, 60, 8
END


#define IDD_ABOUT  999

IDD_ABOUT DIALOG DISCARDABLE  0, 0, 239, 66
STYLE DS_MODALFRAME | WS_POPUP | WS_CAPTION | WS_SYSMENU
CAPTION "About Box"
FONT 8, "MS Sans Serif"
BEGIN
    DEFPUSHBUTTON   "OK", IDOK, 174, 18, 50, 14
    PUSHBUTTON      "Cancel", IDCANCEL, 174, 35, 50, 14
    GROUPBOX        "About this program...", IDC_STATIC, 7, 7, 225, 52
    CTEXT           "A Double Buffering Method Demo\n\nby Angelo Graziosi",
                    IDC_STATIC, 16, 18, 144, 33
END
```

```
!
! (Partial) Fortran Interface to the Windows API Library
! by Angelo Graziosi  (firstname.lastnameATalice.it)
! Copyright Angelo Graziosi
!
! It is distributed in the hope that it will be useful,
! but WITHOUT ANY WARRANTY; without even the implied warranty of
! MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
!
! HOW TO BUILD (MSYS2/MINGW32/MINGW64 shell)
!
!   cd ~/programming/win32−fortran/poisson2D
!
!   rm −rf {*.mod,*.res,~/programming/modules/*} && \
!   windres poisson2D.rc −O coff −o poisson2D.res && \
!   gfortran −O3 −Wall −mwindows −J ~/programming/modules \
!     ~/programming/basic−modules/basic_mods.f90 \
!     ../{win32.f90,win32boxes.f90,win32app.f90} \
!     poisson2D.f90 poisson2D.res −o poisson2D.out && \
!     rm −rf {*.mod,*.res,~/programming/modules/*}
!
! In MINGW32/MINGW64, add '−static' and:
!
!   poisson2D.out ==> poisson2D−mingw32/mingw64
!
!
! DESCRIPTION
!    Boundary Value Problem for Poisson Equation.
!    We solve the Dirichlet problem for Poisson equation in two
!    dimension with overrelaxation of Gauss−Seidel method.
!    The equation is
!
!       Uxx+Uyy = −S(x,y)
!
!    where Uxx (Uyy) is the 2nd partial derivative w.r.t. x (y) of U(x,y),
!    the potential. −S(x,y) is the charge density.
!
! References
!    Press W.H., Numerical Recipes, C.U.P
!    Karlen D., Computational Physics, Carleton University
!    Koonin S.E., Computational Physics, Addison−Wesley
!
! Remember that:
!
!   int(0,UINT_T)    −−>  0_UINT_T
!   int(0,WPARAM_T)  −−>  0_WPARAM_T
!   int(0,LPARAM_T)  −−>  0_LPARAM_T
!

! A new implementation of shade.kumac PAW macro. See
!   http://paw.web.cern.ch/paw/allfaqs.html
module color_map
  use win32, only: COLORREF_T, RGB
  implicit none
  private

  integer, parameter :: NMXPT = 20

  integer, parameter, public :: MAXCOLOURINDEX = 255
  integer, parameter, public :: MAXCOLOURS = MAXCOLOURINDEX+1

  integer(COLORREF_T), public :: crColors(0:MAXCOLOURINDEX)

  integer :: npt, idx(NMXPT)
  integer :: r(NMXPT), g(NMXPT), b(NMXPT)

  public :: set_color_map

contains

  subroutine set_shade(idxi,ri,gi,bi)
    integer, intent(in) :: idxi, ri, gi, bi
    if (idxi < 0) then
       npt = 0
       return
    end if
    npt = npt+1
    if (npt > NMXPT) then
```

```
      write(*,*) 'Error: too many colours'
      stop
    endif
    idx(npt) = idxi
    r(npt)   = ri
    g(npt)   = gi
    b(npt)   = bi
  end subroutine set_shade

  subroutine shade()
    integer :: i, ii, i1, i2, j, n, rs, gs, bs, r1, g1, b1, r2, g2, b2
    real :: scale
    if (npt < 2) then
      write(*,*) 'Error: at least two colours are needed'
      stop
    endif
    do i = 2, npt
      j  = i−1
      i1 = idx(j)
      i2 = idx(i)
      r1 = r(j)
      g1 = g(j)
      b1 = b(j)
      r2 = r(i)
      g2 = g(i)
      b2 = b(i)
      n  = i2−i1+1
      do ii = i1, i2
        scale = (ii−i1)/(n−1.0)
        rs = int((r2 − r1)*scale + r1)
        gs = int((g2 − g1)*scale + g1)
        bs = int((b2 − b1)*scale + b1)

        crColors(ii) = RGB(rs,gs,bs)
      enddo
    enddo
  end subroutine shade

  subroutine set_color_map()
    !crColors(0) = RGB(0,0,0)           ! BLACK
    !crColors(255) = RGB(255,255,255)   ! WHITE

    ! The first call to set_shade() MUST be this INITIALIZATION
    call set_shade( −1,0,0,0)

    call set_shade(  0,  0,  0,128)
    call set_shade( 40,  0,  0,255)
    call set_shade(100,  0,255,255)
    call set_shade(120,  0,255,128)
    call set_shade(160,255,255,  0)
    call set_shade(255,128,  0,  0)
    call shade()
  end subroutine set_color_map
end module color_map

module the_app
  use kind_consts, only: DP
  use color_map
  use AboutBox_class
  use XBox_class
  use XYBox_class
  use RadioBox_class
  use win32, only: BLACK_COLOR, BLACKNESS, COLORREF_T, BOOL_T, DWORD_T, &
       FALSE_T, HBITMAP_T, HBRUSH_T, HDC_T, HINSTANCE_T, HWND_T, IDYES, &
       INT_T, LPARAM_T, LRESULT_T, MAX_FMT, MAX_LEN, NL, NUL, NULL_T, &
       TA_CENTER, TA_LEFT, TRUE_T, UINT_T, VK_ESCAPE, WHITE_COLOR, WM_CHAR, &
       WM_CLOSE, WM_COMMAND, WM_CREATE, WM_DESTROY, WM_SIZE, WORD_T, &
       WPARAM_T, &
       ask_confirmation, CreateCompatibleDC, CreateSolidBrush, &
       DefWindowProc, DeleteDC, DeleteObject, DestroyWindow, error_msg, &
       GetDC, GetStockObject, lo_word, MessageBeep, PostMessage, &
       PostQuitMessage, ReleaseDC, SelectObject, SetBkColor, SetTextAlign, &
       SetTextColor
  use win32app, only: box_type, win32app_BitBlt, win32app_clearDC, &
       win32app_CreateCompatibleBitmap, win32app_fillbox, win32app_setup, &
       win32app_textout, win32app_xmin, win32app_xmax, win32app_ymax
  implicit none
  private
```

```
integer(WORD_T), parameter, public :: IDI_POISSON2D = 1
integer(WORD_T), parameter, public :: IDM_MAINMENU  = 9000

integer(WORD_T), parameter :: IDM_FILE_EXIT       = 9010
integer(WORD_T), parameter :: IDM_DATA_NDIV       = 9020
integer(WORD_T), parameter :: IDM_DATA_MAXI       = 9021
integer(WORD_T), parameter :: IDM_DATA_EPS        = 9022
integer(WORD_T), parameter :: IDM_DATA_OMEGA      = 9023
integer(WORD_T), parameter :: IDM_DATA_XBOUNDS    = 9024
integer(WORD_T), parameter :: IDM_DATA_YBOUNDS    = 9025
integer(WORD_T), parameter :: IDM_DATA_NSOUT      = 9026
integer(WORD_T), parameter :: IDM_DATA_PHILIMITS  = 9027
integer(WORD_T), parameter :: IDM_OPTIONS_CFGTYPE = 9030
integer(WORD_T), parameter :: IDM_OPTIONS_FLDTYPE = 9031
integer(WORD_T), parameter :: IDM_RUNAPP          = 9040
integer(WORD_T), parameter :: IDM_HELP_DISCLAIMER = 9998
integer(WORD_T), parameter :: IDM_HELP_ABOUT      = 9999

!integer(WORD_T), parameter, public :: IDC_STATIC = -1

integer(WORD_T), parameter :: IDD_DATA_NDIV = 100
integer(INT_T), parameter :: IDC_NDIV       = 101

integer(WORD_T), parameter :: IDD_DATA_MAXI = 200
integer(INT_T), parameter :: IDC_MAXI       = 201

integer(WORD_T), parameter :: IDD_DATA_EPS = 300
integer(INT_T), parameter :: IDC_EPS       = 301

integer(WORD_T), parameter :: IDD_DATA_OMEGA = 400
integer(INT_T), parameter :: IDC_OMEGA       = 401

integer(WORD_T), parameter :: IDD_DATA_XBOUNDS = 500
integer(INT_T), parameter :: IDC_ULEFT         = 501
integer(INT_T), parameter :: IDC_URIGHT        = 502

integer(WORD_T), parameter :: IDD_DATA_YBOUNDS = 600
integer(INT_T), parameter :: IDC_UBOTTOM       = 601
integer(INT_T), parameter :: IDC_UTOP          = 602

integer(WORD_T), parameter :: IDD_DATA_NSOUT = 700
integer(INT_T), parameter :: IDC_NSOUT       = 701

integer(WORD_T), parameter :: IDD_DATA_PHILIMITS = 800
integer(INT_T), parameter :: IDC_PHIMIN          = 801
integer(INT_T), parameter :: IDC_PHIMAX          = 802

integer(WORD_T), parameter :: IDD_OPTIONS_CFGTYPE = 900
integer(INT_T), parameter :: IDC_ONEBOX           = 901
integer(INT_T), parameter :: IDC_TWOBOX           = 902
integer(INT_T), parameter :: IDC_CONDENSER        = 903
integer(INT_T), parameter :: IDC_THREECHARGES     = 904
integer(INT_T), parameter :: IDC_CHARGEDLINE      = 905
integer(INT_T), parameter :: ONE_BOX       = 1
integer(INT_T), parameter :: TWO_BOX       = 2
integer(INT_T), parameter :: CONDENSER     = 3
integer(INT_T), parameter :: THREE_CHARGES = 4
integer(INT_T), parameter :: CHARGED_LINE  = 5

integer(WORD_T), parameter :: IDD_OPTIONS_FLDTYPE = 950
integer(INT_T), parameter :: IDC_POTENTIAL        = 951
integer(INT_T), parameter :: IDC_GRADIENT         = 952
integer(INT_T), parameter :: POTENTIAL_FLD = 1
integer(INT_T), parameter :: GRADIENT_FLD  = 2

integer(WORD_T), parameter :: IDD_DISCLAIMER = 998
integer(WORD_T), parameter :: IDD_ABOUT = 999

! COMMON data
integer(HBITMAP_T) :: hBitmap = NULL_T
logical :: run_flag = .true.
real(DP) :: box_xmin, box_xmax, box_ymax

! Application data, strictly speaking...
logical :: converg = .false., not_converg = .true.
integer :: ndiv = 100, max_iter = 100, count_iter = 0, nsout = 10, &
     cfg_type = ONE_BOX, fld_type = POTENTIAL_FLD
```

```
  real(DP) :: eps = 0.0001_DP, omega = 1.8_DP
  real(DP) :: u_left = 1.0_DP, u_right = 0.0_DP, &
       u_bottom = 0.0_DP, u_top = 0.0_DP, &
       phi_min = 0.0_DP, phi_max = 1.0_DP
  real(DP) :: dphi = 0.0_DP, h = 0.0_DP, hq = 0.0_DP, h2 = 0.0_DP, &
       omega1 = 0.0_DP, omega4 = 0.0_DP, &
       energy = 0.0_DP, energy_old = 0.0_DP

  ! Dynamic memory...
  real(DP), target, allocatable :: u(:,:),f(:,:)
  real(DP), allocatable :: s(:,:)
  real(DP), pointer :: phi(:,:) => null()
  logical, allocatable :: b(:,:)

  public :: paint_screen, WndProc

contains

  subroutine grid_on()
    integer :: ierr

    allocate(u(0:ndiv,0:ndiv),stat=ierr)
    if (ierr /= 0) then
       write(*,*) '*** FATAL ERROR ***'
       write(*,*) 'U: Allocation request denied'
       stop
    end if

    allocate(s(0:ndiv,0:ndiv),stat=ierr)
    if (ierr /= 0) then
       write(*,*) '*** FATAL ERROR ***'
       write(*,*) 'S: Allocation request denied'
       stop
    end if

    allocate(f(0:ndiv,0:ndiv),stat=ierr)
    if (ierr /= 0) then
       write(*,*) '*** FATAL ERROR ***'
       write(*,*) 'F: Allocation request denied'
       stop
    end if

    allocate(b(0:ndiv,0:ndiv),stat=ierr)
    if (ierr /= 0) then
       write(*,*) '*** FATAL ERROR ***'
       write(*,*) 'B: Allocation request denied'
       stop
    end if

    ! Now that all is allocated, we can associate the pointer
    if (fld_type == POTENTIAL_FLD) then
       phi => u
    else
       phi => f
    end if
  end subroutine grid_on

  subroutine grid_off()
    integer :: ierr

    if (allocated(b)) deallocate(b,stat=ierr)
    if (ierr /= 0) then
       write(*,*) '*** FATAL ERROR ***'
       write(*,*) 'B: Deallocation request denied'
       stop
    end if

    if (allocated(f)) deallocate(f,stat=ierr)
    if (ierr /= 0) then
       write(*,*) '*** FATAL ERROR ***'
       write(*,*) 'F: Deallocation request denied'
       stop
    end if

    if (allocated(s)) deallocate(s,stat=ierr)
    if (ierr /= 0) then
       write(*,*) '*** FATAL ERROR ***'
       write(*,*) 'S: Deallocation request denied'
```

```
      stop
    end if

    if (allocated(u)) deallocate(u,stat=ierr)
    if (ierr /= 0) then
      write(*,*) '*** FATAL ERROR ***'
      write(*,*) 'U: Deallocation request denied'
      stop
    end if

    ! Now that all is deallocated, we can deassociate the pointer
    nullify(phi)
  end subroutine grid_off

  subroutine fcn_one_box(x,y,u,s,b)
    real(DP), intent(in) :: x, y
    real(DP), intent(out) :: u, s
    logical, intent(out) :: b
    real(DP) :: not_used_x, not_used_y
    not_used_x = x
    not_used_y = y
    u = 0.0_DP
    s = 0.0_DP
    b = .false.
  end subroutine fcn_one_box

  subroutine fcn_two_box(x,y,u,s,b)
    real(DP), intent(in) :: x, y
    real(DP), intent(out) :: u, s
    logical, intent(out) :: b
    ! A box inside a box without charge. The inner box is
    ! (0.25,0.25) − (0.75,0.75), but the boundary conditions are assigned
    ! ONLY on its perimeter, NOT on its inner points!!!
    ! (We have a kind of square ring...)
    integer :: i, i1, i2, j, j1, j2

    i1 = int(0.25_DP/h)
    i2 = int(0.75_DP/h)

    j1 = i1
    j2 = i2

    i = int(x/h)
    j = int(y/h)

    s = 0.0_DP

    if (((i == i1 .or. i == i2) .and. (j1 <= j .and. j <= j2)) .or. &
        ((j == j1 .or. j == j2) .and. (i1 <= i .and. i <= i2))) then
      u = 1.0_DP
      b = .true.
    else
      u = 0.0_DP
      b = .false.
    end if
  end subroutine fcn_two_box

  subroutine fcn_condenser(x,y,u,s,b)
    real(DP), intent(in) :: x, y
    real(DP), intent(out) :: u, s
    logical, intent(out) :: b
    ! A Condenser inside a box without charge
    ! The condenser plates are at 0.25 and 0.75
    integer :: i, i1, i2, j, j1, j2

    i1 = int(0.25_DP/h)
    i2 = int(0.75_DP/h)

    j1 = i1
    j2 = i2

    i = int(x/h)
    j = int(y/h)

    s = 0.0_DP

    if ((i == i1) .and. (j1 <= j .and. j <= j2)) then
      u = 1.0_DP
```

```
      b = .true.
   else if ((i == i2) .and. (j1 <= j .and. j <= j2)) then
      u = −1.0_DP
      b = .true.
   else
      u = 0.0_DP
      b = .false.
   end if
end subroutine fcn_condenser

function delta(x,hwhm)
   real(DP) :: delta
   real(DP), intent(in) :: x, hwhm
   real(DP), parameter :: PI = 3.14159265358979323846_DP
   delta = hwhm/(PI*(hwhm*hwhm+x*x))
end function delta

function theta(x)
   real(DP) :: theta
   real(DP), intent(in) :: x
   if (x > 0.0_DP) then
      theta = 1.0_DP
   else
      theta = 0.0_DP
   end if
end function theta

subroutine fcn_three_charges(x,y,u,s,b)
   real(DP), intent(in) :: x, y
   real(DP), intent(out) :: u, s
   logical, intent(out) :: b
   real(DP), parameter :: HWHM = 0.0005_DP

   u = 0.0_DP
   b = .false.
   s = 0.1_DP*(delta(x−0.25_DP,HWHM)*delta(y−0.25_DP,HWHM) &
        +delta(x−0.75_DP,HWHM)*delta(y−0.25_DP,HWHM) &
        −delta(x−0.5_DP,HWHM)*delta(y−0.75_DP,HWHM))
end subroutine fcn_three_charges

subroutine fcn_charged_line(x,y,u,s,b)
   real(DP), intent(in) :: x, y
   real(DP), intent(out) :: u, s
   logical, intent(out) :: b
   real(DP), parameter :: HWHM = 0.0005_DP
   real(DP) :: not_used_y

   not_used_y = y

   u = 0.0_DP
   b = .false.

   s = delta(x−0.5_DP,HWHM)*(theta(x−0.25_DP)−theta(x−0.75_DP))
end subroutine fcn_charged_line

function get_grid_energy()
   real(DP) :: get_grid_energy
   integer, save :: i, j
   real(DP), save :: sum1, sum2, ff

   ! Koonin’s formula (6.7):
   !
   ! E = 0.5*Sum(i=1,N)Sum(j=1,N)[(u(i,j)−u(i−1,j))**2+(u(i,j)−u(i,j−1))**2]
   !   −h*h*Sum(i=1,N−1)Sum(j=1,N−1)[S(i,j)*u(i,j)]
   !

   sum1 = 0.0_DP
   sum2 = 0.0_DP

   do i = 1, ndiv
      do j = 1, ndiv

         ! The (length of the) gradient
         ff = hypot(u(i,j)−u(i−1,j),u(i,j)−u(i,j−1))

         f(i,j) = ff

         ! The total sum of the gradient squared
```

```
         sum1 = sum1+ff*ff

         ! On i == ndiv, j == ndiv we have s == 0: this means summing
         ! for i = 1,N−1, j = 1,N−1
         sum2 = sum2+s(i,j)*u(i,j)
      end do
   end do

   ! Completing the calculus of the field.
   ! We assume the continuity of the field
   f(1:ndiv,0) = f(1:ndiv,1)
   f(0,1:ndiv) = f(1,1:ndiv)
   f(0,0) = f(1,1)

   ! s(i,j) = hq*S(i*h,j*h)
   get_grid_energy = 0.5_DP*sum1−sum2
end function get_grid_energy

subroutine setup_grid()
   real(DP) :: x, y
   integer :: i, j

   ! Iterations initialization
   count_iter = 0

   ! Grid initialization
   ! The method, as you can see, NEVER uses the values of density s(:,:) on
   ! boundaries! This is used as a trick in computing the energy: we
   ! set s(:,:) on boundary to ZERO.
   ! b(i,j) == true if in x = i*h, y = j*h there is a boundary condition
   !
   ! First the bottom and top side...
   do i = 0, ndiv
      u(i,0) = u_bottom
      u(i,ndiv) = u_top
      s(i,0) = 0.0_DP
      s(i,ndiv) = 0.0_DP
      f(i,0) = 0.0_DP
      f(i,ndiv) = 0.0_DP
      b(i,0) = .true.
      b(i,ndiv) = .true.
   end do

   ! ...then the left and right side...
   do j = 0, ndiv
      u(0,j) = u_left
      u(ndiv,j) = u_right
      s(0,j) = 0.0_DP
      s(ndiv,j) = 0.0_DP
      f(0,j) = 0.0_DP
      f(ndiv,j) = 0.0_DP
      b(0,j) = .true.
      b(ndiv,j) = .true.
   end do

   ! ...then the inner nodes
   do i = 1, ndiv−1
      x = i*h
      do j = 1, ndiv−1
         y = j*h
         select case(cfg_type)
         case(ONE_BOX)
            call fcn_one_box(x,y,u(i,j),s(i,j),b(i,j))
         case(TWO_BOX)
            call fcn_two_box(x,y,u(i,j),s(i,j),b(i,j))
         case(CONDENSER)
            call fcn_condenser(x,y,u(i,j),s(i,j),b(i,j))
         case(THREE_CHARGES)
            call fcn_three_charges(x,y,u(i,j),s(i,j),b(i,j))
         case(CHARGED_LINE)
            call fcn_charged_line(x,y,u(i,j),s(i,j),b(i,j))

         case default
            call fcn_one_box(x,y,u(i,j),s(i,j),b(i,j))
         end select

         s(i,j) = s(i,j)*hq
         f(i,j) = 0.0_DP
```

```
      end do
    end do

    ! Initialization of energy and convergence flags
    energy = get_grid_energy()
    converg = .false.
    not_converg = .true.
  end subroutine setup_grid

  subroutine draw_colorbar(hdc)
    integer(HDC_T), intent(in) :: hdc
    !
    ! A simple colour bar scale
    !
    character(len=*), parameter :: NAME_FLD(2) = [ &
         'Potential', &
         'Gradient ' ]
    integer, parameter :: SCALE_PTS = 5
    integer(COLORREF_T), save :: old_bk_color, old_text_color
    integer(UINT_T), save :: old_align
    integer(HBRUSH_T), save :: hBrush
    character(len=MAX_LEN), save :: buffer = ''
    integer, save :: i
    real(DP), save :: bar_width, delta_y, phi, delta_phi
    type(box_type), save :: bar_box
    integer, save :: dummy

    ! The space on the right of the grid is in [1,box_xmax], the bar width is
    ! 1/15
    bar_width = (box_xmax−1.0_DP)/15

    ! In X, the bar is in [1+7*bar_width,1+8*bar_width], i.e at position 8
    ! (7+1+7 = 15)
    bar_box%x1 = 1.0_DP+7.0_DP*bar_width
    bar_box%x2 = 1.0_DP+8.0_DP*bar_width

    ! In Y the bar is in [0,1] and composed of 0, 1,... MAXCOLOURINDEX
    ! filled slices
    delta_y = (1.0_DP−0.0_DP) / MAXCOLOURS

    bar_box%y2 = 0.0_DP
    do i = 0, MAXCOLOURINDEX
       bar_box%y1 = bar_box%y2
       bar_box%y2 = bar_box%y2+delta_y

       hBrush = CreateSolidBrush(crColors(i))
       dummy = int(SelectObject(hdc,hBrush),INT_T)

       dummy = win32app_fillbox(hdc,bar_box,hBrush)

       dummy = DeleteObject(hBrush)
    end do

    delta_phi = (phi_max−phi_min)/(SCALE_PTS−1)
    delta_y = (1.0_DP−0.0_DP)/(SCALE_PTS−1)

    old_bk_color = SetBkColor(hdc,BLACK_COLOR)
    old_text_color = SetTextColor(hdc,WHITE_COLOR)
    old_align = SetTextAlign(hdc,TA_LEFT)

    phi = phi_min  ! U
    bar_box%x2 = bar_box%x2+0.2_DP*bar_width
    bar_box%y2 = 0.0_DP+0.2_DP*bar_width
    do i = 1, SCALE_PTS
       buffer = ''
       write(buffer,'(f10.4)') phi
       buffer = trim(adjustl(buffer))//'   '//NUL
       dummy = win32app_textout(hdc,bar_box%x2,bar_box%y2,buffer)

       phi = phi+delta_phi
       bar_box%y2 = bar_box%y2+delta_y
    end do

    bar_box%x2 = bar_box%x2+0.8_DP*bar_width
    bar_box%y2 = bar_box%y2−3*delta_y/2

    buffer = ''
    write(buffer,*) NAME_FLD(fld_type)
```

```
  buffer = trim(adjustl(buffer))//'   '//NUL
  dummy = win32app_textout(hdc,bar_box%x2,bar_box%y2,buffer)

  ! Restore previous text colors...
  dummy = SetTextAlign(hdc,old_align)
  dummy = SetTextColor(hdc,old_text_color)
  dummy = SetBkColor(hdc,old_bk_color)
end subroutine draw_colorbar

subroutine draw_grid(hdc)
  integer(HDC_T), intent(in) :: hdc
  character(len=*), parameter :: FMT = '(a,i6,2(a,1pg14.7),a,i6,a,1pg14.7)'
  ! We use SAVE just to save something at each call
  ! (draw_time() is called intensively, at each iteration)
  integer(COLORREF_T), save :: old_bk_color, old_text_color
  integer(UINT_T), save :: old_align
  integer(HBRUSH_T), save :: hBrush
  character(len=MAX_LEN), save :: buffer = ''
  type(box_type), save :: box
  integer, save :: i, j, i_col
  real(DP), save :: x, y
  integer, save :: dummy

  old_bk_color = SetBkColor(hdc,BLACK_COLOR)
  old_text_color = SetTextColor(hdc,WHITE_COLOR)
  old_align = SetTextAlign(hdc,TA_CENTER)

  x = 0.5_DP
  y = 0.5_DP*(1.0_DP+box_ymax)
  buffer = ''
  write(buffer,FMT) 'NDIV = ',ndiv,'   OMEGA = ',omega,'   EPS = ',eps, &
       '   COUNT = ',count_iter,'   E = ',energy
  buffer = trim(adjustl(buffer))//'   '//NUL
  dummy = win32app_textout(hdc,x,y,buffer)

  ! Restore previous text colors...
  dummy = SetTextAlign(hdc,old_align)
  dummy = SetTextColor(hdc,old_text_color)
  dummy = SetBkColor(hdc,old_bk_color)

  x = −h2
  box%x2 = x
  do i = 0, ndiv
     box%x1 = box%x2

     x = x+h
     box%x2 = x

     y = −h2
     box%y2 = y
     do j = 0, ndiv
        box%y1 = box%y2

        y = y+h
        box%y2 = y

        i_col = int((phi(i,j)−phi_min)/dphi)

        if (i_col < 0) then
           i_col = 0
        else if (i_col > MAXCOLOURINDEX) then
           i_col = MAXCOLOURINDEX
        end if

        hBrush = CreateSolidBrush(crColors(i_col))
        dummy = int(SelectObject(hdc,hBrush),INT_T)

        dummy = win32app_fillbox(hdc,box,hBrush)

        dummy = DeleteObject(hBrush)
     end do
  end do
end subroutine draw_grid

subroutine painting_setup(hWnd)
  integer(HWND_T), intent(in) :: hWnd
  logical, save :: first = .true.
  integer(HDC_T) :: hdc, hdcMem
```

```
   integer :: dummy

   if (first) then
      call set_color_map()
      dphi = (phi_max−phi_min)/MAXCOLOURS
      h = 1.0_DP/ndiv
      hq = h*h
      h2 = 0.5_DP*h
      omega1 = 1.0_DP−omega
      omega4 = 0.25_DP*omega
      call setup_grid()
      first = .false.
   end if

   if (hBitmap /= NULL_T) then
      dummy = DeleteObject(hBitmap)
   end if

   hdc = GetDC(hWnd)
   hdcMem = CreateCompatibleDC(hdc)
   hBitmap = win32app_CreateCompatibleBitmap(hdc)
   dummy = ReleaseDC(hWnd,hdc)

   dummy = int(SelectObject(hdcMem,hBitmap),INT_T)

   ! Clear the off−screen DC (hdcMem) for the next drawing
   dummy = win32app_clearDC(hdcMem,BLACKNESS)

   ! Draw (on the off−screen DC) the color scale
   call draw_colorbar(hdcMem)

   ! Draw (on the off−screen DC) grid at current iteration
   call draw_grid(hdcMem)

   dummy = DeleteDC(hdcMem)
end subroutine painting_setup

function process_char(hWnd,wParam)
   integer(BOOL_T) :: process_char
   integer(HWND_T), intent(in) :: hWnd
   integer(WPARAM_T), intent(in) :: wParam
   integer :: dummy

   select case(wParam)
   case(VK_ESCAPE)
      dummy = DestroyWindow(hWnd)

      process_char = TRUE_T
      return

   case default
      process_char = FALSE_T
      return
   end select
end function process_char

subroutine set_ndiv(hWnd)
   integer(HWND_T), intent(in) :: hWnd
   character(len=MAX_FMT), parameter :: FMT = '(f12.0)'
   type(XBox) :: xb
   real(DP) :: x

   x = real(ndiv,DP)

   call new_box(xb,hWnd,IDD_DATA_NDIV,IDC_NDIV,FMT,x)

   if (run(xb) > 0) then

      x = get(xb)

      if (x > 0.0_DP) then
         ! Destroy the current grid...
         call grid_off()

         ! Get the new value
         ndiv = int(x)

         ! Readjust some params
```

```
         h = 1.0_DP/ndiv
         hq = h*h
         h2 = 0.5_DP*h

         ! Create the new grid...
         call grid_on()

         ! If you prefer to see something, uncomment the following...
         !call setup_grid()
      else
         call error_msg('NDIV <= 0 !!!'//NL &
              //'Must be NDIV > 0...   '//NUL)
      end if
   end if
end subroutine set_ndiv

subroutine set_maxi(hWnd)
   integer(HWND_T), intent(in) :: hWnd
   character(len=MAX_FMT), parameter :: FMT = '(f12.0)'
   type(XBox) :: xb
   real(DP) :: x

   x = real(max_iter,DP)

   call new_box(xb,hWnd,IDD_DATA_MAXI,IDC_MAXI,FMT,x)

   if (run(xb) > 0) then

      x = get(xb)

      if (x > 0.0_DP) then
         max_iter = int(x)
      else
         call error_msg('MAX_ITER <= 0 !!!'//NL &
              //'Must be MAX_ITER > 0...   '//NUL)
      end if
   end if
end subroutine set_maxi

subroutine set_eps(hWnd)
   integer(HWND_T), intent(in) :: hWnd
   character(len=MAX_FMT), parameter :: FMT = '(1pg12.5)'
   type(XBox) :: xb
   real(DP) :: x

   x = eps

   call new_box(xb,hWnd,IDD_DATA_EPS,IDC_EPS,FMT,x)

   if (run(xb) > 0) then

      x = get(xb)

      if (0.0_DP < x .and. x < 1.0_DP) then
         eps = x
      else
         call error_msg('EPS not in (0,1)!!!'//NL &
              //'Must be 0 < EPS < 1...   '//NUL)
      end if
   end if
end subroutine set_eps

subroutine set_omega(hWnd)
   integer(HWND_T), intent(in) :: hWnd
   character(len=MAX_FMT), parameter :: FMT = '(1pg12.5)'
   type(XBox) :: xb
   real(DP) :: x

   x = omega

   call new_box(xb,hWnd,IDD_DATA_OMEGA,IDC_OMEGA,FMT,x)

   if (run(xb) > 0) then

      x = get(xb)

      if (0.0_DP < x .and. x < 2.0_DP) then
         omega = x
```

```
         ! Readjust some params
         omega1 = 1.0_DP-omega
         omega4 = 0.25_DP*omega
      else
         call error_msg('OMEGA not in (0,2)!!!'//NL &
              //'Must be 0 < OMEGA < 2...   '//NUL)
      end if
   end if
end subroutine set_omega

subroutine set_xbounds(hWnd)
   integer(HWND_T), intent(in) :: hWnd
   character(len=MAX_FMT), parameter :: FMT = '(1pg12.5)'
   type(XYBox) :: xyb

   call new_box(xyb,hWnd,IDD_DATA_XBOUNDS,IDC_ULEFT,IDC_URIGHT,FMT, &
        u_left,u_right)

   if (run(xyb) > 0) then
      u_left = get_x(xyb)
      u_right = get_y(xyb)
   end if
end subroutine set_xbounds

subroutine set_ybounds(hWnd)
   integer(HWND_T), intent(in) :: hWnd
   character(len=MAX_FMT), parameter :: FMT = '(1pg12.5)'
   type(XYBox) :: xyb

   call new_box(xyb,hWnd,IDD_DATA_YBOUNDS,IDC_UBOTTOM,IDC_UTOP,FMT, &
        u_bottom,u_top)

   if (run(xyb) > 0) then
      u_bottom = get_x(xyb)
      u_top = get_y(xyb)
   end if
end subroutine set_ybounds

subroutine set_nsout(hWnd)
   integer(HWND_T), intent(in) :: hWnd
   character(len=MAX_FMT), parameter :: FMT = '(f12.0)'
   type(XBox) :: xb
   real(DP) :: x

   x = real(nsout,DP)

   call new_box(xb,hWnd,IDD_DATA_NSOUT,IDC_NSOUT,FMT,x)

   if (run(xb) > 0) then

      x = get(xb)

      if (x > 0.0_DP) then
         nsout = int(x)
      else
         call error_msg('NSOUT <= 0 !!!'//NL &
              //'Must be NSOUT > 0...   '//NUL)
      end if
   end if
end subroutine set_nsout

subroutine set_phi_limits(hWnd)
   integer(HWND_T), intent(in) :: hWnd
   character(len=MAX_FMT), parameter :: FMT = '(1pg12.5)'
   type(XYBox) :: xyb
   real(DP) :: x, y

   x = phi_min
   y = phi_max

   call new_box(xyb,hWnd,IDD_DATA_PHILIMITS,IDC_PHIMIN,IDC_PHIMAX,FMT,x,y)

   if (run(xyb) > 0) then

      x = get_x(xyb)
      y = get_y(xyb)
```

```
      if (x < y) then
         phi_min = x
         phi_max = y

         ! Readjust some params
         dphi = (phi_max−phi_min)/MAXCOLOURS
      else
         call error_msg('PHI_MIN >= PHI_MAX !!!'//NL &
              //'Must be PHI_MIN < PHI_MAX...   '//NUL)
      end if
   end if
end subroutine set_phi_limits

subroutine set_cfgtype(hWnd)
   integer(HWND_T), intent(in) :: hWnd
   integer, parameter :: NUM_BUTTONS = 5
   character(len=*), parameter :: BUTTON_NAMES(NUM_BUTTONS) = [ &
        '&One Box      ', &
        '&Two Box      ', &
        '&Condenser    ', &
        'T&hree Charges', &
        'Charged &Line ' ]
   type(RadioBox) :: rb

   call new_box(rb,hWnd,IDD_OPTIONS_CFGTYPE,IDC_ONEBOX,BUTTON_NAMES, &
        NUM_BUTTONS,cfg_type)

   if (run(rb) > 0) cfg_type = get_current_button(rb)
end subroutine set_cfgtype

subroutine set_fldtype(hWnd)
   integer(HWND_T), intent(in) :: hWnd
   integer, parameter :: NUM_BUTTONS = 2
   character(len=*), parameter :: BUTTON_NAMES(NUM_BUTTONS) = [ &
        '&Potential', &
        '&Gradient ' ]
   type(RadioBox) :: rb

   call new_box(rb,hWnd,IDD_OPTIONS_FLDTYPE,IDC_POTENTIAL,BUTTON_NAMES, &
        NUM_BUTTONS,fld_type)

   if (run(rb) > 0) then
      fld_type = get_current_button(rb)

      ! Now we can re−associate the pointer
      if (fld_type == POTENTIAL_FLD) then
         phi => u
      else
         phi => f
      end if
   end if
end subroutine set_fldtype

subroutine disclaimer_dlg(hWnd)
   integer(HWND_T), intent(in) :: hWnd
   type(AboutBox) :: ab
   integer :: dummy
   call new_box(ab,hWnd,IDD_DISCLAIMER)
   dummy = run(ab)
end subroutine disclaimer_dlg

subroutine about_dlg(hWnd)
   integer(HWND_T), intent(in) :: hWnd
   type(AboutBox) :: ab
   integer :: dummy
   call new_box(ab,hWnd,IDD_ABOUT)
   dummy = run(ab)
end subroutine about_dlg

function process_command(hWnd,wParam)
   integer(BOOL_T) :: process_command
   integer(HWND_T), intent(in) :: hWnd
   integer(WPARAM_T), intent(in) :: wParam
   integer :: dummy

   run_flag = .false.

   select case(lo_word(int(wParam,DWORD_T)))
```

```
case(IDM_FILE_EXIT)
   dummy = MessageBeep(64)
   if (ask_confirmation(hWnd,'Sure you want to exit?   '//NUL, &
        'Exit?'//NUL) == IDYES) then
      dummy = PostMessage(hWnd,WM_CLOSE,0_WPARAM_T,0_LPARAM_T)
   end if
   process_command = TRUE_T
   return

case(IDM_DATA_NDIV)
   call set_ndiv(hWnd)
   process_command = TRUE_T
   return

case(IDM_DATA_MAXI)
   call set_maxi(hWnd)
   process_command = TRUE_T
   return

case(IDM_DATA_EPS)
   call set_eps(hWnd)
   process_command = TRUE_T
   return

case(IDM_DATA_OMEGA)
   call set_omega(hWnd)
   process_command = TRUE_T
   return

case(IDM_DATA_XBOUNDS)
   call set_xbounds(hWnd)
   process_command = TRUE_T
   return

case(IDM_DATA_YBOUNDS)
   call set_ybounds(hWnd)
   process_command = TRUE_T
   return

case(IDM_DATA_NSOUT)
   call set_nsout(hWnd)
   process_command = TRUE_T
   return

case(IDM_DATA_PHILIMITS)
   call set_phi_limits(hWnd)
   process_command = TRUE_T
   return

case(IDM_OPTIONS_CFGTYPE)
   call set_cfgtype(hWnd)
   process_command = TRUE_T
   return

case(IDM_OPTIONS_FLDTYPE)
   call set_fldtype(hWnd)
   process_command = TRUE_T
   return

case(IDM_RUNAPP)
   run_flag = .true.
   call setup_grid()
   process_command = TRUE_T
   return

case(IDM_HELP_DISCLAIMER)
   call disclaimer_dlg(hWnd)
   process_command = TRUE_T
   return

case(IDM_HELP_ABOUT)
   call about_dlg(hWnd)
   process_command = TRUE_T
   return

case default
   process_command = FALSE_T
   return
```

```
  end select
end function process_command

function WndProc(hWnd,iMsg,wParam,lParam) bind(C)
  !GCC$ ATTRIBUTES STDCALL :: WndProc
  integer(LRESULT_T) :: WndProc
  integer(HWND_T), value :: hWnd
  integer(UINT_T), value :: iMsg
  integer(WPARAM_T), value :: wParam
  integer(LPARAM_T), value :: lParam

  logical, save :: first = .true.
  integer :: dummy

  select case(iMsg)
  case(WM_CREATE)
     ! Creating the grid...
     call grid_on()

     WndProc = 0
     return

  case(WM_SIZE)
     if (first) then
        call win32app_setup(lParam,−0.7_DP,1.7_DP,−0.5_DP,1.5_DP)
        first = .false.
     else
        call win32app_setup(lParam)
     end if

     ! Getting the box boundaries... each time, maybe, the mapping changed...
     box_xmin = win32app_xmin()
     box_xmax = win32app_xmax()
     box_ymax = win32app_ymax()

     ! ...and initialize the painting
     call painting_setup(hWnd)

     WndProc = 0
     return

  case(WM_CHAR)
     if (process_char(hWnd,wParam) == TRUE_T) then
        WndProc = 0
        return
     end if
     ! ...else it continues with DefWindowProc

  case(WM_COMMAND)
     if (process_command(hWnd,wParam) == TRUE_T) then
        WndProc = 0
        return
     end if
     ! ...else it continues with DefWindowProc

  case(WM_CLOSE)
     dummy = DestroyWindow(hWnd)
     WndProc = 0
     return

  case(WM_DESTROY)
     if (hBitmap /= NULL_T) then
        dummy = DeleteObject(hBitmap)
     end if

     ! Destroying the grid...
     call grid_off()

     call PostQuitMessage(0)
     ! Commenting out the next two statements, it continues
     ! with DefWindowProc()
     WndProc = 0
     return
  end select

  WndProc = DefWindowProc(hWnd,iMsg,wParam,lParam)
end function WndProc
```

```
  subroutine update_grid()
    ! We use SAVE just to save something at each call
    ! (update_ball_position() is called intensively, at each iteration)
    integer, save :: i, j

    do i = 1, ndiv−1
       do j = 1, ndiv−1
          if (b(i,j)) cycle
          u(i,j) = omega1*u(i,j) &
               +omega4*(u(i+1,j)+u(i−1,j)+u(i,j+1)+u(i,j−1)+s(i,j))
       end do
    end do
  end subroutine update_grid

  subroutine paint_screen(hWnd)
    integer(HWND_T), intent(in) :: hWnd
    ! We use SAVE just to save something at each call
    ! (paint_screen() is called intensively, at each iteration)
    integer(HDC_T), save :: hdc, hdcMem
    integer, save :: dummy

    if (hBitmap /= NULL_T) then
       hdc = GetDC(hWnd)
       hdcMem = CreateCompatibleDC(hdc)
       dummy = int(SelectObject(hdcMem,hBitmap),INT_T)

       if ((mod(count_iter,nsout) == 0) .or. converg) then
          ! Transfer the off−screen DC to the screen
          dummy = win32app_BitBlt(hdc,hdcMem)
       end if

       dummy = ReleaseDC(hWnd,hdc)

       if (count_iter < max_iter .and. not_converg .and. run_flag) then
          energy_old = energy        ! Save current energy
          count_iter = count_iter+1  ! Update iteration counter...

          call update_grid()         ! ...then UPDATE the grid
          energy = get_grid_energy() ! Get the energy for the updated grid

          ! Set the convergence flags for the updated grid
          not_converg = abs(energy−energy_old) > eps
          converg = .not. not_converg
       end if

       if ((mod(count_iter,nsout) == 0) .or. converg) then
          ! Clear the off−screen DC (hdcMem) for the next drawing
          dummy = win32app_clearDC(hdcMem,BLACKNESS)

          ! Draw (on the off−screen DC) the color scale
          call draw_colorbar(hdcMem)

          ! Draw (on the off−screen DC) grid at current iteration
          call draw_grid(hdcMem)
       end if

       dummy = DeleteDC(hdcMem)
    end if
  end subroutine paint_screen
end module the_app

function WinMain(hInstance,hPrevInstance,lpCmdLine,nCmdShow) &
     bind(C, name='WinMain')
  use, intrinsic :: iso_c_binding, only: C_PTR, C_CHAR, c_sizeof, c_funloc, &
       C_FUNPTR, c_loc
  use win32, only: BLACK_BRUSH, CS_HREDRAW, CS_VREDRAW, CW_USEDEFAULT, &
       DWORD_T, HINSTANCE_T, HWND_T, INT_T, NUL, NULL_PTR_T, NULL_T, &
       PM_REMOVE, WM_QUIT, WS_OVERLAPPEDWINDOW, UINT_T, &
       MSG_T, WNDCLASSEX_T, &
       arrow_cursor, CreateWindowEx, DispatchMessage, error_msg, ExitProcess, &
       GetStockObject, LoadCursor, LoadIcon, make_int_resource, &
       make_int_resource_C_PTR, PeekMessage, RegisterClassEx, ShowWindow, &
       TranslateMessage, UpdateWindow
  use the_app, only: IDI_POISSON2D, IDM_MAINMENU, &
       paint_screen, WndProc
  implicit none
  !GCC$ ATTRIBUTES STDCALL :: WinMain
  integer(INT_T) :: WinMain
```

```
  integer(HINSTANCE_T),  value :: hInstance
  integer(HINSTANCE_T),  value :: hPrevInstance
  type(C_PTR), value :: lpCmdLine ! LPSTR
  integer(INT_T), value :: nCmdShow

  character(kind=C_CHAR,len=128), target :: app_name = &
       'Poisson2D'//NUL
  character(kind=C_CHAR,len=*), parameter :: WINDOW_CAPTION = &
       'Poisson Equation in 2D'//NUL
  type(WNDCLASSEX_T) :: WndClass
  integer(HWND_T) :: hWnd
  type(MSG_T) :: msg
  integer :: dummy

  ! To avoid some annoying warnings at compile time...
  integer(HINSTANCE_T) :: not_used_hPrevInstance
  type(C_PTR) :: not_used_lpCmdLine
  not_used_hPrevInstance = hPrevInstance
  not_used_lpCmdLine = lpCmdLine

  WndClass%cbSize = int(c_sizeof(Wndclass),UINT_T)
  WndClass%style = ior(CS_HREDRAW,CS_VREDRAW)
  WndClass%lpfnWndProc = c_funloc(WndProc)
  WndClass%cbClsExtra = 0
  WndClass%cbWndExtra = 0
  WndClass%hInstance = hInstance
  WndClass%hIcon = LoadIcon(hInstance,make_int_resource(IDI_POISSON2D))
  WndClass%hCursor = LoadCursor(NULL_T,arrow_cursor())
  WndClass%hbrBackground = GetStockObject(BLACK_BRUSH)
  WndClass%lpszMenuName = make_int_resource_C_PTR(IDM_MAINMENU)
  WndClass%lpszClassName = c_loc(app_name(1:1))
  WndClass%hIconSm = LoadIcon(hInstance,make_int_resource(IDI_POISSON2D))

  if (RegisterClassEx(WndClass) == 0) then
     call error_msg('Window Registration Failure!   '//NUL)
     call ExitProcess(0_UINT_T)
     WinMain = 0
     !return
  end if

  hWnd = CreateWindowEx(0_DWORD_T, &
       app_name, &
       WINDOW_CAPTION, &
       WS_OVERLAPPEDWINDOW, &
       CW_USEDEFAULT,CW_USEDEFAULT,CW_USEDEFAULT,CW_USEDEFAULT, &
       NULL_T,NULL_T,hInstance,NULL_PTR_T)

  if (hWnd == NULL_T) then
     call error_msg('Window Creation Failure!   '//NUL)
     call ExitProcess(0_UINT_T)
     WinMain = 0
     !return
  end if

  dummy = ShowWindow(hWnd,nCmdShow)
  dummy = UpdateWindow(hWnd)

  ! See: Charles Petzold "Programming Windows", 5th ed., pag. 162
  ! 'Random Rectangles'
  do
     if (PeekMessage(msg,NULL_T,0,0,PM_REMOVE) /= 0) then
        if (msg%message == WM_QUIT) exit
        dummy = TranslateMessage(msg)
        dummy = int(DispatchMessage(msg),INT_T)
     else
        call paint_screen(hWnd)
     end if
  end do

  call ExitProcess(int(msg%wParam,UINT_T))
  WinMain = 0
end function WinMain
```

```
//
// (Partial) Fortran Interface to the Windows API Library
// by Angelo Graziosi  (firstname.lastnameATalice.it)
// Copyright Angelo Graziosi
//
// It is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
//
// RC file for "poisson2D" app
//

#define IDI_POISSON2D 1

IDI_POISSON2D ICON DISCARDABLE "../common_icons/smiling_sun.ico"

#define IDM_MAINMENU        9000
#define IDM_FILE_EXIT       9010
#define IDM_DATA_NDIV       9020
#define IDM_DATA_MAXI       9021
#define IDM_DATA_EPS        9022
#define IDM_DATA_OMEGA      9023
#define IDM_DATA_XBOUNDS    9024
#define IDM_DATA_YBOUNDS    9025
#define IDM_DATA_NSOUT      9026
#define IDM_DATA_PHILIMITS  9027
#define IDM_OPTIONS_CFGTYPE 9030
#define IDM_OPTIONS_FLDTYPE 9031
#define IDM_RUNAPP          9040
#define IDM_HELP_DISCLAIMER 9998
#define IDM_HELP_ABOUT      9999

IDM_MAINMENU MENU DISCARDABLE
BEGIN
    POPUP "&File"
    BEGIN
        MENUITEM "E&xit...", IDM_FILE_EXIT
    END

    POPUP "&Data"
    BEGIN
        MENUITEM  "Number of lattice &divisions...", IDM_DATA_NDIV
        MENUITEM  "Maximum number of &iterations...", IDM_DATA_MAXI
        MENUITEM  "&Precision for convergence...", IDM_DATA_EPS
        MENUITEM  "&Relaxation parameter...", IDM_DATA_OMEGA
        MENUITEM SEPARATOR
        MENUITEM  "&X−boundary conditions...", IDM_DATA_XBOUNDS
        MENUITEM  "&Y−boundary conditions...", IDM_DATA_YBOUNDS
        MENUITEM SEPARATOR
        MENUITEM  "Iteration steps for &output...", IDM_DATA_NSOUT
        MENUITEM  "Color &scale limits...", IDM_DATA_PHILIMITS
    END

    POPUP "&Options"
    BEGIN
        MENUITEM  "&Problem solving...", IDM_OPTIONS_CFGTYPE
        MENUITEM  "Plotting &field...", IDM_OPTIONS_FLDTYPE
    END

    POPUP "&Run Application"
    BEGIN
        MENUITEM "R&un", IDM_RUNAPP
    END

    POPUP "&Help"
    BEGIN
        MENUITEM "&Disclaimer...", IDM_HELP_DISCLAIMER
        MENUITEM "&About...", IDM_HELP_ABOUT
    END
END

#include <windows.h>

#define IDC_STATIC −1

#define IDC_CTEXT "The Dirichlet problem for Poisson equation in 2D."

#define IDD_DATA_NDIV  100
```

```
#define IDC_NDIV       101

IDD_DATA_NDIV DIALOG DISCARDABLE  0, 0, 284, 77
STYLE DS_MODALFRAME | WS_POPUP | WS_CAPTION | WS_SYSMENU
CAPTION "Number of lattice divisions"
FONT 8, "MS Sans Serif"
BEGIN
    DEFPUSHBUTTON   "OK", IDOK, 227, 7, 50, 14
    PUSHBUTTON      "Cancel", IDCANCEL, 227, 24, 50, 14
    CTEXT           IDC_CTEXT, IDC_STATIC, 7, 7, 153, 18
    GROUPBOX        "Number of lattice &divisions", IDC_STATIC, 13, 30, 186, 34
    EDITTEXT        IDC_NDIV, 35, 43, 60, 14, ES_AUTOHSCROLL
    LTEXT           " ;−) ", IDC_STATIC, 97, 45, 20, 8
END

#define IDD_DATA_MAXI  200
#define IDC_MAXI       201

IDD_DATA_MAXI DIALOG DISCARDABLE  0, 0, 284, 77
STYLE DS_MODALFRAME | WS_POPUP | WS_CAPTION | WS_SYSMENU
CAPTION "Maximum number of iterations"
FONT 8, "MS Sans Serif"
BEGIN
    DEFPUSHBUTTON   "OK", IDOK, 227, 7, 50, 14
    PUSHBUTTON      "Cancel", IDCANCEL, 227, 24, 50, 14
    CTEXT           IDC_CTEXT, IDC_STATIC, 7, 7, 153, 18
    GROUPBOX        "Maximum number of &iterations",
                    IDC_STATIC, 13, 30, 186, 34
    EDITTEXT        IDC_MAXI, 35, 43, 60, 14, ES_AUTOHSCROLL
    LTEXT           " ;−) ", IDC_STATIC, 97, 45, 20, 8
END

#define IDD_DATA_EPS  300
#define IDC_EPS       301

IDD_DATA_EPS DIALOG DISCARDABLE  0, 0, 284, 77
STYLE DS_MODALFRAME | WS_POPUP | WS_CAPTION | WS_SYSMENU
CAPTION "Precision for convergence"
FONT 8, "MS Sans Serif"
BEGIN
    DEFPUSHBUTTON   "OK", IDOK, 227, 7, 50, 14
    PUSHBUTTON      "Cancel", IDCANCEL, 227, 24, 50, 14
    CTEXT           IDC_CTEXT, IDC_STATIC, 7, 7, 153, 18
    GROUPBOX        "&Precision for convergence", IDC_STATIC, 13, 30, 186, 34
    EDITTEXT        IDC_EPS, 35, 43, 60, 14, ES_AUTOHSCROLL
    LTEXT           " ;−) ", IDC_STATIC, 97, 45, 20, 8
END

#define IDD_DATA_OMEGA  400
#define IDC_OMEGA       401

IDD_DATA_OMEGA DIALOG DISCARDABLE  0, 0, 284, 77
STYLE DS_MODALFRAME | WS_POPUP | WS_CAPTION | WS_SYSMENU
CAPTION "Relaxation parameter"
FONT 8, "MS Sans Serif"
BEGIN
    DEFPUSHBUTTON   "OK", IDOK, 227, 7, 50, 14
    PUSHBUTTON      "Cancel", IDCANCEL, 227, 24, 50, 14
    CTEXT           IDC_CTEXT, IDC_STATIC, 7, 7, 153, 18
    GROUPBOX        "&Relaxation parameter", IDC_STATIC, 13, 30, 186, 34
    EDITTEXT        IDC_OMEGA, 35, 43, 60, 14, ES_AUTOHSCROLL
    LTEXT           " ;−) ", IDC_STATIC, 97, 45, 20, 8
END

#define IDD_DATA_XBOUNDS  500
#define IDC_ULEFT         501
#define IDC_URIGHT        502

IDD_DATA_XBOUNDS DIALOG DISCARDABLE  0, 0, 284, 97
STYLE DS_MODALFRAME | WS_POPUP | WS_CAPTION | WS_SYSMENU
CAPTION "X−boundary conditions"
FONT 8, "MS Sans Serif"
BEGIN
    DEFPUSHBUTTON   "OK", IDOK, 227, 7, 50, 14
    PUSHBUTTON      "Cancel", IDCANCEL, 227, 24, 50, 14
    CTEXT           IDC_CTEXT, IDC_STATIC, 7, 7, 153, 18
    GROUPBOX        "&X−boundary conditions", IDC_STATIC, 13, 30, 186, 54
    LTEXT           "U(&L) : ", IDC_STATIC, 35, 45, 30, 8
```

```
    EDITTEXT        IDC_ULEFT, 85, 43, 60, 14, ES_AUTOHSCROLL
    LTEXT           " ;−) ", IDC_STATIC, 147, 45, 20, 8
    LTEXT           "U(&R) : ", IDC_STATIC, 35, 65, 30, 8
    EDITTEXT        IDC_URIGHT, 85, 63, 60, 14, ES_AUTOHSCROLL
    LTEXT           " ;−) ", IDC_STATIC, 147, 65, 20, 8
END

#define IDD_DATA_YBOUNDS  600
#define IDC_UBOTTOM       601
#define IDC_UTOP          602

IDD_DATA_YBOUNDS DIALOG DISCARDABLE  0, 0, 284, 97
STYLE DS_MODALFRAME | WS_POPUP | WS_CAPTION | WS_SYSMENU
CAPTION "Y−boundary conditions"
FONT 8, "MS Sans Serif"
BEGIN
    DEFPUSHBUTTON   "OK", IDOK, 227, 7, 50, 14
    PUSHBUTTON      "Cancel", IDCANCEL, 227, 24, 50, 14
    CTEXT           IDC_CTEXT, IDC_STATIC, 7, 7, 153, 18
    GROUPBOX        "&Y−boundary conditions", IDC_STATIC, 13, 30, 186, 54
    LTEXT           "U(&B) : ", IDC_STATIC, 35, 45, 30, 8
    EDITTEXT        IDC_UBOTTOM, 85, 43, 60, 14, ES_AUTOHSCROLL
    LTEXT           " ;−) ", IDC_STATIC, 147, 45, 20, 8
    LTEXT           "U(&T) : ", IDC_STATIC, 35, 65, 30, 8
    EDITTEXT        IDC_UTOP, 85, 63, 60, 14, ES_AUTOHSCROLL
    LTEXT           " ;−) ", IDC_STATIC, 147, 65, 20, 8
END

#define IDD_DATA_NSOUT  700
#define IDC_NSOUT       701

IDD_DATA_NSOUT DIALOG DISCARDABLE  0, 0, 284, 77
STYLE DS_MODALFRAME | WS_POPUP | WS_CAPTION | WS_SYSMENU
CAPTION "Iteration steps for output"
FONT 8, "MS Sans Serif"
BEGIN
    DEFPUSHBUTTON   "OK", IDOK, 227, 7, 50, 14
    PUSHBUTTON      "Cancel", IDCANCEL, 227, 24, 50, 14
    CTEXT           IDC_CTEXT, IDC_STATIC, 7, 7, 153, 18
    GROUPBOX        "Iteration steps for &output", IDC_STATIC, 13, 30, 186, 34
    EDITTEXT        IDC_NSOUT, 35, 43, 60, 14, ES_AUTOHSCROLL
    LTEXT           " ;−) ", IDC_STATIC, 97, 45, 20, 8
END

#define IDD_DATA_PHILIMITS  800
#define IDC_PHIMIN          801
#define IDC_PHIMAX          802

IDD_DATA_PHILIMITS DIALOG DISCARDABLE  0, 0, 284, 97
STYLE DS_MODALFRAME | WS_POPUP | WS_CAPTION | WS_SYSMENU
CAPTION "Color scale limits"
FONT 8, "MS Sans Serif"
BEGIN
    DEFPUSHBUTTON   "OK", IDOK, 227, 7, 50, 14
    PUSHBUTTON      "Cancel", IDCANCEL, 227, 24, 50, 14
    CTEXT           IDC_CTEXT, IDC_STATIC, 7, 7, 153, 18
    GROUPBOX        "Color &scale limits", IDC_STATIC, 13, 30, 186, 54
    LTEXT           "M&IN : ", IDC_STATIC, 35, 45, 30, 8
    EDITTEXT        IDC_PHIMIN, 85, 43, 60, 14, ES_AUTOHSCROLL
    LTEXT           " ;−) ", IDC_STATIC, 147, 45, 20, 8
    LTEXT           "M&AX : ", IDC_STATIC, 35, 65, 30, 8
    EDITTEXT        IDC_PHIMAX, 85, 63, 60, 14, ES_AUTOHSCROLL
    LTEXT           " ;−) ", IDC_STATIC, 147, 65, 20, 8
END

#define IDD_OPTIONS_CFGTYPE  900
#define IDC_ONEBOX           901
#define IDC_TWOBOX           902
#define IDC_CONDENSER        903
#define IDC_THREECHARGES     904
#define IDC_CHARGEDLINE      905

IDD_OPTIONS_CFGTYPE DIALOG DISCARDABLE  0, 0, 284, 157
STYLE DS_MODALFRAME | WS_POPUP | WS_CAPTION | WS_SYSMENU
CAPTION "Problem solving"
FONT 8, "MS Sans Serif"
BEGIN
    DEFPUSHBUTTON   "OK", IDOK, 227, 7, 50, 14
```

```
    PUSHBUTTON      "Cancel", IDCANCEL, 227, 24, 50, 14
    CTEXT           IDC_CTEXT, IDC_STATIC, 7, 7, 153, 18
    GROUPBOX        "&Problem solving", IDC_STATIC, 13, 30, 186, 114
    RADIOBUTTON     "&One Box", IDC_ONEBOX, 35, 45, 60, 8
    RADIOBUTTON     "&Two Box", IDC_TWOBOX, 35, 65, 60, 8
    RADIOBUTTON     "&Condenser", IDC_CONDENSER, 35, 85, 60, 8
    RADIOBUTTON     "T&hree Charges", IDC_THREECHARGES, 35, 105, 60, 8
    RADIOBUTTON     "Charged &Line", IDC_CHARGEDLINE, 35, 125, 60, 8
END

#define IDD_OPTIONS_FLDTYPE  950
#define IDC_POTENTIAL        951
#define IDC_GRADIENT         952

IDD_OPTIONS_FLDTYPE DIALOG DISCARDABLE  0, 0, 284, 97
STYLE DS_MODALFRAME | WS_POPUP | WS_CAPTION | WS_SYSMENU
CAPTION "Plotting field"
FONT 8, "MS Sans Serif"
BEGIN
    DEFPUSHBUTTON   "OK", IDOK, 227, 7, 50, 14
    PUSHBUTTON      "Cancel", IDCANCEL, 227, 24, 50, 14
    CTEXT           IDC_CTEXT, IDC_STATIC, 7, 7, 153, 18
    GROUPBOX        "Plotting &field", IDC_STATIC, 13, 30, 186, 54
    RADIOBUTTON     "&Potential", IDC_POTENTIAL, 35, 45, 60, 8
    RADIOBUTTON     "&Gradient", IDC_GRADIENT, 35, 65, 60, 8
END

#define IDD_DISCLAIMER  998

IDD_DISCLAIMER DIALOG DISCARDABLE  0, 0, 319, 66
STYLE DS_MODALFRAME | WS_POPUP | WS_CAPTION | WS_SYSMENU
CAPTION "Disclaimer Box"
FONT 8, "MS Sans Serif"
BEGIN
    DEFPUSHBUTTON   "OK", IDOK, 254, 18, 50, 14
    PUSHBUTTON      "Cancel", IDCANCEL, 254, 35, 50, 14
    GROUPBOX        "Disclaimer...", IDC_STATIC, 7, 7, 305, 52
    CTEXT           "It is distributed in the hope that it will be useful,\n"
                    "but WITHOUT ANY WARRANTY; without even the implied "
                    "warranty of\n"
                    "MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.",
                    IDC_STATIC, 16, 18, 224, 33
END


#define IDD_ABOUT  999

IDD_ABOUT DIALOG DISCARDABLE  0, 0, 239, 66
STYLE DS_MODALFRAME | WS_POPUP | WS_CAPTION | WS_SYSMENU
CAPTION "About Box"
FONT 8, "MS Sans Serif"
BEGIN
    DEFPUSHBUTTON   "OK", IDOK, 174, 18, 50, 14
    PUSHBUTTON      "Cancel", IDCANCEL, 174, 35, 50, 14
    GROUPBOX        "About this program...", IDC_STATIC, 7, 7, 225, 52
    CTEXT           "A Solution for Poisson equation.\n\n"
                    "by (C) Angelo Graziosi",
                    IDC_STATIC, 16, 18, 144, 33
END
```