W I N B G I M − 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.

On September 2006 we started writing fortran modules which partially, if not totally, interface WinBGIm−3.6 library. This is a modern C/C++ re−implementation (using Windows API) of the Borland Graphics Interface.

Here we present the modules (f03bgi.f90, gks_bgi.f90) and an example of application (cobra_bgi.f90). As always, details in the comments.

```
! ============================================================================
!
! Fortran Interface to the WinBGIm-3.6 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.
!
!
! Created     : Sep 15, 2006
! Last change : May 14, 2015
!
! NOTE 0
!   Instead of adding 'use, intrinsic :: iso_c_binding' in each procedure
!   interface, use 'import' as in the 'f03bgi' module.
!
! NOTE 1
!    Attention should be put on array declarations.
!    In C/C++ the array index starts from 0 while in Fortran it starts from
!    1, so if an array expects an index variable or a constant from 0 it
!    cannot be declared: integer :: v(MAXDIM) but MUST v(0:MAXDIM-1).
!    For example we could have
!
!       do i = BLACK,WHITE   ...pal&color(i) ... enddo
!
!    so 'color()' in palettetype must be decalred color(0:15) and NOT
!    color(16).
!
! NOTE 2
!    The attribute 'value' belongs to F2003 standard
!    For the BOZ (binary, octal, hexadecimal) constants, we need the
!    conversion with int() function. For example:
!
!       RED_VALUE = (iand((v),int(Z'FF')))
!
!    Indeed, F2003 'thinks' that "Z'FF" is an integer(8) (64 bit, on Win32)
!    or integer(16) (128 bit, on GNU Linux K10.04) and complains about the
!    implicit conversion.
!    The above happens with GFortran-4.6 .GE. 20100528.
!
! MODULES
!
!       f03bgi_types
!       f03bgi
!
! ============================================================================

module f03bgi_types
  use kind_consts
  use, intrinsic :: iso_c_binding
  implicit none
  integer, parameter :: MAXBUF = 256
  ! ==================
  !  COLORS constants
  ! ==================
  integer, parameter :: MAXCOLORS = 15
  enum, bind(C)
     enumerator :: BLACK,&
          BLUE,GREEN,CYAN,RED,MAGENTA,&
          BROWN,LIGHTGRAY,DARKGRAY,LIGHTBLUE,LIGHTGREEN,&
          LIGHTCYAN,LIGHTRED,LIGHTMAGENTA,YELLOW,WHITE
  end enum
  ! ======================
  !  WRITE_MODES constants
  ! ======================
  enum, bind(C)
     enumerator :: COPY_PUT,&
          XOR_PUT,OR_PUT,AND_PUT,NOT_PUT
  end enum
  ! ======================
  !  LINE_STYLES constants
  ! ======================
  enum, bind(C)
     enumerator :: SOLID_LINE,&
          DOTTED_LINE,CENTER_LINE,DASHED_LINE,USERBIT_LINE
  end enum
```

```
! ======================
!  FILL_STYLES constants
! ======================
enum, bind(C)
   enumerator :: EMPTY_FILL,&
         SOLID_FILL,LINE_FILL,LTSLASH_FILL,SLASH_FILL,&
         BKSLASH_FILL,LTBKSLASH_FILL,HATCH_FILL,XHATCH_FILL,&
         INTERLEAVE_FILL,WIDE_DOT_FILL,CLOSE_DOT_FILL,USER_FILL
end enum
! ==========================
!  TEXT_DIRECTIONS constants
! ==========================
enum, bind(C)
   enumerator :: HORIZ_DIR,VERT_DIR
end enum
! ======================
!  FONT_TYPES constants
! ======================
enum, bind(C)
   enumerator :: DEFAULT_FONT,&
         TRIPLEX_FONT,SMALL_FONT,SANSSERIF_FONT,GOTHIC_FONT,SCRIPT_FONT,&
         SIMPLEX_FONT,TRIPLEXSCRIPT_FONT,COMPLEX_FONT,EUROPEAN_FONT,BOLD_FONT
end enum
! ========================
!  Text justify constants
! ========================
enum, bind(C)
   enumerator :: LEFT_TEXT,&
         CENTER_TEXT,RIGHT_TEXT
   enumerator :: BOTTOM_TEXT = 0,TOP_TEXT = 2
end enum
! ==========================
!  Line thickness constants
! ==========================
enum, bind(C)
   enumerator :: NORM_WIDTH = 1,THICK_WIDTH = 3
end enum
! ======================
!  Others line constants
! ======================
enum, bind(C)
   enumerator :: DOTTEDLINE_LENGTH = 2,CENTRELINE_LENGTH = 4
   enumerator :: USER_CHAR_SIZE = 0
end enum
! ============================
!  Viewport clipping constants
! ============================
enum, bind(C)
   enumerator :: CLIP_ON = 1,CLIP_OFF = 0
   enumerator :: TOP_ON = 1,TOP_OFF = 0
end enum
! ===========================================================================
!  Definitions for the key pad extended keys are added here. I have also
!  modified getch() so that when one of these keys are pressed, getch will
!  return a zero followed by one of these values. This is the same way
!  that it works in conio for dos applications.
! ===========================================================================
enum, bind(C)
   enumerator :: KEY_HOME = 71,KEY_UP,KEY_PGUP
   enumerator :: KEY_LEFT = 75,KEY_CENTER,KEY_RIGHT
   enumerator :: KEY_END = 79,KEY_DOWN,KEY_PGDN,KEY_INSERT,KEY_DELETE
   enumerator :: KEY_F1 = 59,KEY_F2,KEY_F3,KEY_F4,KEY_F5,KEY_F6,KEY_F7,&
         KEY_F8,KEY_F9
end enum
! ==========================
!  GRAPHICS_ERRORS constants
! ==========================
enum, bind(C)
   enumerator :: grOk = 0,&
         grNoInitGraph = −1,&
         grNotDetected = −2,&
         grFileNotFound = −3,&
         grInvalidDriver   = −4,&
         grNoLoadMem = −5,&
         grNoScanMem = −6,&
         grNoFloodMem = −7,&
         grFontNotFound = −8,&
         grNoFontMem = −9,&
```

```
        grInvalidMode =   −10,&
        grError = −11,&
        grIOerror = −12,&
        grInvalidFont = −13,&
        grInvalidFontNum = −14,&
        grInvalidDeviceNum = −15,&
        grInvalidVersion = −18
end enum
! ========================================================================
!  Graphics drivers constants, includes X11 which is particular to XBGI.
! ========================================================================
enum, bind(C)
   enumerator :: DETECT,&
        CGA,MCGA,EGA,EGA64,EGAMONO,&
        IBM8514,HERCMONO,ATT400,VGA,PC3270
end enum
! ==========================
!  Graphics modes constants
! ==========================
enum, bind(C)
   enumerator :: CGAC0 = 0,CGAC1,CGAC2,CGAC3,CGAHI
   enumerator :: MCGAC0 = 0,MCGAC1,MCGAC2,MCGAC3,MCGAMED,MCGAHI
   enumerator :: EGALO = 0,EGAHI = 1
   enumerator :: EGA64LO = 0,EGA64HI = 1,EGAMONOHI = 3
   enumerator :: HERCMONOHI = 0
   enumerator :: ATT400C0 = 0,ATT400C1,ATT400C2,ATT400C3,ATT400MED,ATT400HI
   enumerator :: VGALO = 0,VGAMED,VGAHI,VGAMAX
   enumerator :: PC3270HI = 0
   enumerator :: IBM8514LO = 0,IBM8514HI
end enum
! ====================================
!  Kind parameters for mouse functions
!  From /usr/include/w32api/winuser.h
! ====================================
enum, bind(C)
   enumerator :: WM_MOUSEMOVE = 512,&
     WM_LBUTTONDBLCLK = 515,&
     WM_LBUTTONDOWN = 513,&
     WM_LBUTTONUP = 514,&
     WM_MBUTTONDBLCLK = 521,&
     WM_MBUTTONDOWN = 519,&
     WM_MBUTTONUP = 520,&
     WM_RBUTTONDOWN = 516,&
     WM_RBUTTONUP = 517,&
     WM_RBUTTONDBLCLK = 518
end enum
! ====================================================================
!  Virtual−Key Codes
!  From:
!  http://msdn.microsoft.com/en−us/library/dd375731%28v=VS.85%29.aspx
! ====================================================================
enum, bind(C)
   enumerator :: VK_LBUTTON = Z'01',VK_RBUTTON = Z'02',&
        VK_CANCEL = Z'03',&
        VK_MBUTTON = Z'04',&
        VK_XBUTTON1 = Z'05',VK_XBUTTON2 = Z'06',&
        VK_BACK = Z'08',&
        VK_TAB = Z'09',&
        VK_CLEAR = Z'0C',&
        VK_RETURN = Z'0D',&
        VK_SHIFT = Z'10',&
        VK_CONTROL = Z'11',&
        VK_MENU = Z'12',&
        VK_PAUSE = Z'13',&
        VK_CAPITAL = Z'14',&
        VK_KANA = Z'15',&
        VK_HANGUEL = Z'15',&
        VK_HANGUL = Z'15',&
        VK_JUNJA = Z'17',&
        VK_FINAL = Z'18',&
        VK_HANJA = Z'19',&
        VK_KANJI = Z'19',&
        VK_ESCAPE = Z'1B',&
        VK_CONVERT = Z'1C',&
        VK_NONCONVERT = Z'1D',&
        VK_ACCEPT = Z'1E',&
        VK_MODECHANGE = Z'1F',&
        VK_SPACE = Z'20',&
```

```
          VK_PRIOR = Z'21',&
          VK_NEXT = Z'22',&
          VK_END = Z'23',&
          VK_HOME = Z'24',&
          VK_LEFT = Z'25',&
          VK_UP = Z'26',&
          VK_RIGHT = Z'27',&
          VK_DOWN = Z'28',&
          VK_SELECT = Z'29',&
          VK_PRINT = Z'2A',&
          VK_EXECUTE = Z'2B',&
          VK_SNAPSHOT = Z'2C',&
          VK_INSERT = Z'2D',&
          VK_DELETE = Z'2E',&
          VK_HELP = Z'2F'
  end enum
  type, bind(C) :: arccoordstype
     integer(C_INT) :: x,y,xstart,ystart,xend,yend
  end type arccoordstype
  type, bind(C) :: fillsettingstype
     integer(C_INT) :: pattern,color
  end type fillsettingstype
  type, bind(C) :: linesettingstype
     integer(C_INT) :: linestyle,upattern,thickness
  end type linesettingstype
  type, bind(C) :: palettetype
     integer(C_SIGNED_CHAR) :: size,colors(0:15)
  end type palettetype
  type, bind(C) :: textsettingstype
     integer(C_INT) :: font,direction,charsize,horiz,vert
  end type textsettingstype
  type, bind(C) :: viewporttype
     integer(C_INT) :: left,top,right,bottom,clip
  end type viewporttype
end module f03bgi_types

module f03bgi
  use, intrinsic :: iso_c_binding
  use f03bgi_types
  implicit none
  interface
     subroutine arc(x,y,stangle,endangle,radius) bind(C)
       import
       integer(C_INT), intent(in), value :: x,y,stangle,endangle,radius
     end subroutine arc
  end interface
  interface
     subroutine bar(left,top,right,bottom) bind(C)
       import
       integer(C_INT), intent(in), value :: left,top,right,bottom
     end subroutine bar
  end interface
  interface
     subroutine bar3d(left,top,right,bottom,depth,topflag) bind(C)
       import
       integer(C_INT), intent(in), value :: left,top,right,bottom,depth,topflag
     end subroutine bar3d
  end interface
  interface
     subroutine circle(x,y,radius) bind(C)
       import
       integer(C_INT), intent(in), value :: x,y,radius
     end subroutine circle
  end interface
  interface
     subroutine cleardevice() bind(C)
       import
     end subroutine cleardevice
  end interface
  interface
     subroutine clearviewport() bind(C)
       import
     end subroutine clearviewport
  end interface
  interface
     subroutine clearmouseclick(kind) bind(C)
       import
       integer(C_INT), intent(in), value :: kind
```

```
   end subroutine clearmouseclick
end interface
interface
   subroutine closegraph() bind(C)
     import
   end subroutine closegraph
end interface
interface
   subroutine delay(millisec) bind(C)
     import
     integer(C_INT), intent(in), value :: millisec
   end subroutine delay
end interface
interface
   subroutine detectgraph(graphdriver,graphmode) bind(C)
     import
     integer(C_INT), intent(out) :: graphdriver,graphmode
   end subroutine detectgraph
end interface
interface
   subroutine drawpoly(numpoints,polypoints) bind(C)
     import
     integer(C_INT), intent(in), value :: numpoints
     ! polypoints should be an array of 2*numpoints elements
     integer(C_INT), intent(in) :: polypoints(*)
   end subroutine drawpoly
end interface
interface
   subroutine ellipse(x,y,stangle,endangle,xradius,yradius) bind(C)
     import
     integer(C_INT), intent(in), value :: x,y,stangle,endangle,&
          xradius,yradius
   end subroutine ellipse
end interface
interface
   subroutine fillellipse(x,y,xradius,yradius) bind(C)
     import
     integer(C_INT), intent(in), value :: x,y,xradius,yradius
   end subroutine fillellipse
end interface
interface
   subroutine fillpoly(numpoints,polypoints) bind(C)
     import
     integer(C_INT), intent(in), value :: numpoints
     ! polypoints should be an array of 2*numpoints elements
     integer(C_INT), intent(in) :: polypoints(*)
   end subroutine fillpoly
end interface
interface
   subroutine floodfill(x,y,border) bind(C)
     import
     integer(C_INT), intent(in), value :: x,y,border
   end subroutine floodfill
end interface
interface
   function getactivepage() bind(C)
     import
     integer(C_INT) :: getactivepage
   end function getactivepage
end interface
interface
   subroutine getarccoords(arccoords) bind(C)
     import
     type (arccoordstype), intent(out) :: arccoords
   end subroutine getarccoords
end interface
interface
   subroutine getaspectratio(xasp,yasp) bind(C)
     import
     integer(C_INT), intent(out) :: xasp,yasp
   end subroutine getaspectratio
end interface
interface
   function getbkcolor() bind(C)
     import
     integer(C_INT) :: getbkcolor
   end function getbkcolor
end interface
```

```
interface
   ! ============================================
   !  This function WORKS only in 'graphics mode'!
   !  See handle_input() C++ source
   ! ============================================
   function getch() bind(C)
     import
     integer(C_INT) :: getch
   end function getch
end interface
interface
   function getcolor() bind(C)
     import
     integer(C_INT) :: getcolor
   end function getcolor
end interface
interface
   function getdefaultpalette() bind(C)
     import
     type (C_PTR) :: getdefaultpalette
   end function getdefaultpalette
end interface
interface
   function getdrivername() bind(C)
     import
     type (C_PTR) :: getdrivername
   end function getdrivername
end interface
interface
   subroutine getfillpattern(pattern) bind(C)
     import
     character(C_CHAR), intent(out) :: pattern(8)
   end subroutine getfillpattern
end interface
interface
   subroutine getfillsettings(fillinfo) bind(C)
     import
     type (fillsettingstype), intent(out) :: fillinfo
   end subroutine getfillsettings
end interface
interface
   function getgraphmode() bind(C)
     import
     integer(C_INT) :: getgraphmode
   end function getgraphmode
end interface
interface
   ! ============================================================
   !  Calling this routine we should pass the address of bitmap:
   !     call getimage(left,top,right,bottom,c_loc(bitmap))
   ! ============================================================
   subroutine getimage(left,top,right,bottom,bitmap) bind(C)
     import
     integer(C_INT), intent(in), value :: left,top,right,bottom
     type (C_PTR), value :: bitmap
   end subroutine getimage
end interface
interface
   subroutine getlinesettings(lineinfo) bind(C)
     import
     type (linesettingstype), intent(out) :: lineinfo
   end subroutine getlinesettings
end interface
interface
   function getmaxcolor() bind(C)
     import
     integer(C_INT) :: getmaxcolor
   end function getmaxcolor
end interface
interface
   function getmaxmode() bind(C)
     import
     integer(C_INT) :: getmaxmode
   end function getmaxmode
end interface
interface
   function getmaxx() bind(C)
     import
```

```
      integer(C_INT) :: getmaxx
    end function getmaxx
  end interface
  interface
    function getmaxy() bind(C)
      import
      integer(C_INT) :: getmaxy
    end function getmaxy
  end interface
  interface
    function getmodename(mode_number) bind(C)
      import
      type (C_PTR) :: getmodename
      integer(C_INT), intent(in), value :: mode_number
    end function getmodename
  end interface
  interface
    subroutine getmoderange(graphdriver,lomode,himode) bind(C)
      import
      integer(C_INT), intent(in), value :: graphdriver
      integer(C_INT), intent(out) :: lomode,himode
    end subroutine getmoderange
  end interface
  interface
    subroutine getmouseclick(kind,x,y) bind(C)
      import
      integer(C_INT), intent(in), value :: kind
      integer(C_INT), intent(out) :: x,y
    end subroutine getmouseclick
  end interface
  interface
    subroutine getpalette(palette) bind(C)
      import
      type (palettetype), intent(out) :: palette
    end subroutine getpalette
  end interface
  interface
    function getpalettesize() bind(C)
      import
      integer(C_INT) :: getpalettesize
    end function getpalettesize
  end interface
  interface
    function getpixel(x,y) bind(C)
      import
      integer(C_INT) :: getpixel
      integer(C_INT), intent(in), value :: x,y
    end function getpixel
  end interface
  interface
    subroutine gettextsettings(texttypeinfo) bind(C)
      import
      type (textsettingstype), intent(out) :: texttypeinfo
    end subroutine gettextsettings
  end interface
  interface
    subroutine getviewsettings(v) bind(C)
      import
      type (viewporttype), intent(out) :: v
    end subroutine getviewsettings
  end interface
  interface
    function getvisualpage() bind(C)
      import
      integer(C_INT) :: getvisualpage
    end function getvisualpage
  end interface
  interface
    function getx() bind(C)
      import
      integer(C_INT) :: getx
    end function getx
  end interface
  interface
    function gety() bind(C)
      import
      integer(C_INT) :: gety
    end function gety
```

```
end interface
interface
   subroutine graphdefaults() bind(C)
     import
   end subroutine graphdefaults
end interface
interface
   function grapherrormsg(errorcode) bind(C)
     import
     type (C_PTR) :: grapherrormsg
     integer(C_INT), intent(in), value :: errorcode
   end function grapherrormsg
end interface
interface
   function graphresult() bind(C)
     import
     integer(C_INT) :: graphresult
   end function graphresult
end interface
interface
   function imagesize(left,top,right,bottom) bind(C)
     import
     integer(C_INT) :: imagesize
     integer(C_INT), intent(in), value :: left,top,right,bottom
   end function imagesize
end interface
interface
   subroutine initgraph(graphdriver,graphmode,pathtodriver) bind(C)
     import
     integer(C_INT), intent(inout) :: graphdriver,graphmode
     !character(C_CHAR), dimension(*), intent(in) :: pathtodriver
     character(C_CHAR), intent(in) :: pathtodriver(*)
   end subroutine initgraph
end interface
interface
   subroutine initwindow(width,height,title,left,top) bind(C)
     import
     integer(C_INT), intent(in), value :: width,height
     character(C_CHAR), intent(in) :: title(*)
     integer(C_INT), intent(in), value :: left,top
   end subroutine initwindow
end interface
interface init_window
   module procedure initwindow2,initwindow3,initwindow4,initwindow5
end interface init_window
! interface
!    ! ============================================================
!    !  This routine is not implemented in the C++ version of BGI
!    !  'detect' is a function pointer, see (get/put)image
!    ! ============================================================
!    function installuserdriver(name,detect) bind(C)
!      import
!      integer(C_INT) :: installuserdriver
!      character(C_CHAR), intent(in) :: name(*)
!      integer(C_INT), intent(in), value :: detect
!    end function installuserdriver
! end interface
! interface
!    ! ============================================================
!    !  This routine is not implemented in the C++ version of BGI
!    ! ============================================================
!    function installuserfont(name) bind(C)
!      import
!      integer(C_INT) :: installuserfont
!      character(C_CHAR), intent(in) :: name(*)
!    end function installuserfont
! end interface
interface
   function is_key(k) bind(C)
     import
     integer(C_INT) :: is_key
     integer(C_INT), intent(in), value :: k
   end function is_key
end interface
interface
   function ismouseclick(kind) bind(C)
     import
     logical(C_BOOL) :: ismouseclick
```

```
      integer(C_INT), intent(in), value :: kind
    end function ismouseclick
  end interface
  interface
    function kbhit() bind(C)
      import
      integer(C_INT) :: kbhit
    end function kbhit
  end interface
  interface
    subroutine line(x1,y1,x2,y2) bind(C)
      import
      integer(C_INT), intent(in), value :: x1,y1,x2,y2
    end subroutine line
  end interface
  interface
    subroutine linerel(dx,dy) bind(C)
      import
      integer(C_INT), intent(in), value :: dx,dy
    end subroutine linerel
  end interface
  interface
    subroutine lineto(x,y) bind(C)
      import
      integer(C_INT), intent(in), value :: x,y
    end subroutine lineto
  end interface
  interface
    function mousex() bind(C)
      import
      integer(C_INT) :: mousex
    end function mousex
  end interface
  interface
    function mousey() bind(C)
      import
      integer(C_INT) :: mousey
    end function mousey
  end interface
  interface
    subroutine moverel(dx,dy) bind(C)
      import
      integer(C_INT), intent(in), value :: dx,dy
    end subroutine moverel
  end interface
  interface
    subroutine moveto(x,y) bind(C)
      import
      integer(C_INT), intent(in), value :: x,y
    end subroutine moveto
  end interface
  interface
    subroutine outtext(textstring) bind(C)
      import
      character(C_CHAR), intent(in) :: textstring(*)
    end subroutine outtext
  end interface
  interface
    subroutine outtextxy(x,y,textstring) bind(C)
      import
      integer(C_INT), intent(in), value :: x,y
      character(C_CHAR), intent(in) :: textstring(*)
    end subroutine outtextxy
  end interface
  interface
    subroutine pieslice(x,y,stangle,endangle,radius) bind(C)
      import
      integer(C_INT), intent(in), value :: x,y,stangle,endangle,radius
    end subroutine pieslice
  end interface
  interface
    ! ============================================================
    !  Calling this routine we should pass the address of bitmap:
    !     call putimage(left,top,c_loc(bitmap),op)
    ! ============================================================
    subroutine putimage(left,top,bitmap,op) bind(C)
      import
      integer(C_INT), intent(in), value :: left,top,op
```

```
      type (C_PTR), value :: bitmap
    end subroutine putimage
  end interface
  interface
    subroutine putpixel(x,y,color) bind(C)
      import
      integer(C_INT), intent(in), value :: x,y,color
    end subroutine putpixel
  end interface
  interface
    subroutine rectangle(left,top,right,bottom) bind(C)
      import
      integer(C_INT), intent(in), value :: left,top,right,bottom
    end subroutine rectangle
  end interface
  ! interface
  !     ! ============================================================
  !     !  This routine is not implemented in the C++ version of BGI
  !     !  'driver' is a function pointer, see (get/put)image
  !     ! ============================================================
  !     function registerbgidriver(driver) bind(C)
  !       import
  !       integer(C_INT) :: registerbgidriver
  !       integer(C_INT), intent(in), value :: driver
  !     end function registerbgidriver
  ! end interface
  ! interface
  !     ! ============================================================
  !     !  This routine is not implemented in the C++ version of BGI
  !     !  'font' is a function pointer, see (get/put)image
  !     ! ============================================================
  !     function registerbgifont(font) bind(C)
  !       import
  !       integer(C_INT) :: registerbgifont
  !       integer(C_INT), intent(in), value :: font
  !     end function registerbgifont
  ! end interface
  interface
     ! ================================================
     !  'h' is a function pointer, see (get/put)image
     !  In C 'h' is 'void h(int,int)'
     ! ================================================
     subroutine registermousehandler(kind,h) bind(C)
       import
       integer(C_INT), intent(in), value :: kind
       type (C_FUNPTR), value :: h
     end subroutine registermousehandler
  end interface
  interface
     ! ==============================
     !  Really it is a dummy routine
     ! ==============================
     subroutine restorecrtmode() bind(C)
       import
     end subroutine restorecrtmode
  end interface
  interface
     subroutine sector(x,y,stangle,endangle,xradius,yradius) bind(C)
       import
       integer(C_INT), intent(in), value :: x,y,stangle,endangle,&
            xradius,yradius
     end subroutine sector
  end interface
  interface
     subroutine setactivepage(page) bind(C)
       import
       integer(C_INT), intent(in), value :: page
     end subroutine setactivepage
  end interface
  interface
     subroutine setallpalette(palette) bind(C)
       import
       type (palettetype), intent(in) :: palette
     end subroutine setallpalette
  end interface
  interface
     subroutine setaspectratio(xasp,yasp) bind(C)
       import
```

```
      integer(C_INT), intent(in), value :: xasp,yasp
    end subroutine setaspectratio
  end interface
  interface
    subroutine setbkcolor(color) bind(C)
      import
      integer(C_INT), intent(in), value :: color
    end subroutine setbkcolor
  end interface
  interface
    subroutine setcolor(color) bind(C)
      import
      integer(C_INT), intent(in), value :: color
    end subroutine setcolor
  end interface
  interface
    subroutine setfillpattern(pattern,color) bind(C)
      import
      character(C_CHAR), intent(in) :: pattern(8)
      integer(C_INT), intent(in), value :: color
    end subroutine setfillpattern
  end interface
  interface
    subroutine setfillstyle(pattern,color) bind(C)
      import
      integer(C_INT), intent(in), value :: pattern,color
    end subroutine setfillstyle
  end interface
  ! interface
  !     ! ============================================================
  !     !  This routine is not implemented in the C++ version of BGI
  !     ! ============================================================
  !     function setgraphbufsize(bufsize) bind(C)
  !       import
  !       integer(C_INT) :: setgraphbufsize
  !       integer(C_INT), intent(in), value :: bufsize
  !     end function setgraphbufsize
  ! end interface
  interface
    ! ==============================
    !  Really it is a dummy routine
    ! ==============================
    subroutine setgraphmode(mode) bind(C)
      import
      integer(C_INT), intent(in), value :: mode
    end subroutine setgraphmode
  end interface
  interface
    subroutine setlinestyle(linestyle,upattern,thickness) bind(C)
      import
      integer(C_INT), intent(in), value :: linestyle,upattern,thickness
    end subroutine setlinestyle
  end interface
  interface
    ! ================================================
    !  This routine does not work as the original in BGI
    ! ================================================
    subroutine setpalette(colornum,color) bind(C)
      import
      integer(C_INT), intent(in), value :: colornum,color
    end subroutine setpalette
  end interface
  interface
    ! ================================================
    !  This routine does not work as the original in BGI
    ! ================================================
    subroutine setrgbpalette(colornum,red,green,blue) bind(C)
      import
      integer(C_INT), intent(in), value :: colornum,red,green,blue
    end subroutine setrgbpalette
  end interface
  interface
    subroutine settextjustify(horiz,vert) bind(C)
      import
      integer(C_INT), intent(in), value :: horiz,vert
    end subroutine settextjustify
  end interface
  interface
```

```
    subroutine settextstyle(font,direction,charsize) bind(C)
      import
      integer(C_INT), intent(in), value :: font,direction,charsize
    end subroutine settextstyle
  end interface
  interface
    subroutine setusercharsize(multx,divx,multy,divy) bind(C)
      import
      integer(C_INT), intent(in), value :: multx,divx,multy,divy
    end subroutine setusercharsize
  end interface
  interface
    subroutine setviewport(left,top,right,bottom,clip) bind(C)
      import
      integer(C_INT), intent(in), value :: left,top,right,bottom,clip
    end subroutine setviewport
  end interface
  interface
    subroutine setvisualpage(page) bind(C)
      import
      integer(C_INT), intent(in), value :: page
    end subroutine setvisualpage
  end interface
  interface
    subroutine setwritemode(mode) bind(C)
      import
      integer(C_INT), intent(in), value :: mode
    end subroutine setwritemode
  end interface
  interface strlen
    ! ====================================================
    !  From Tobias Burnus,
    !  http://gcc.gnu.org/ml/fortran/2010-02/msg00029.html
    !
    !  Note: as both strlen and strlen2 have the same
    !  binding name, you can only use one of them at a
    !  time.
    ! ====================================================
    ! function strlen(str) bind(C)
    !   import
    !   character(kind=C_CHAR) :: str(*)
    !   integer(C_SIZE_T) :: strlen
    ! end function strlen
    function strlen2(str) bind(C,name="strlen")
      import
      type (C_PTR), value :: str
      integer(C_SIZE_T)  :: strlen2
    end function strlen2
  end interface strlen
  interface
    function textheight(textstring) bind(C)
      import
      integer(C_INT) :: textheight
      character(C_CHAR), intent(in) :: textstring(*)
    end function textheight
  end interface
  interface
    function textwidth(textstring) bind(C)
      import
      integer(C_INT) :: textwidth
      character(C_CHAR), intent(in) :: textstring(*)
    end function textwidth
  end interface
contains
  function RGB(r,g,b)
    integer :: RGB
    integer, intent(in) :: r,g,b
    RGB = (ior(ior((r),ishft((g),8)),ishft((b),16)))
  end function RGB
  function IS_BGI_COLOR(c)
    use f03bgi_types
    logical :: IS_BGI_COLOR
    integer, intent(in) :: c
    IS_BGI_COLOR = (((c) >= 0).and.((c) <= MAXCOLORS))
  end function IS_BGI_COLOR
  function IS_RGB_COLOR(c)
    logical :: IS_RGB_COLOR
    integer, intent(in) :: c
```

```
    IS_RGB_COLOR = .false.
    ! ========================================================================
    !  In C a variable is false if its numeric value is NULL, i.e. 0 (ZERO)
    !  It is true if its numeric value is NON−NULL, i.e. < 0 or > 0
    ! ========================================================================
    if ((iand((c),Z'04000000')) /= 0) IS_RGB_COLOR = .true.
  end function IS_RGB_COLOR
  function RED_VALUE(v)
    integer :: RED_VALUE
    integer, intent(in) :: v
    RED_VALUE = (iand((v),int(Z'FF')))
  end function RED_VALUE
  function GREEN_VALUE(v)
    integer :: GREEN_VALUE
    integer, intent(in) :: v
    ! we need to shift right
    GREEN_VALUE = (iand(ishft((v),−8),int(Z'FF')))
  end function GREEN_VALUE
  function BLUE_VALUE(v)
    integer :: BLUE_VALUE
    integer, intent(in) :: v
    ! we need shift right
    BLUE_VALUE = (iand(ishft((v),−16),int(Z'FF')))
  end function BLUE_VALUE
  function COLOR(r,g,b)
    integer :: COLOR
    integer, intent(in) :: r,g,b
    COLOR = (ior(int(Z'04000000'),RGB(r,g,b)))
  end function COLOR
  function RGB_COLOR(c)
    integer :: RGB_COLOR
    integer, intent(in) :: c
    RGB_COLOR = iand(c,int(Z'FFFFFF'))
  end function RGB_COLOR
  function CString(string) result(array)
    character(len=*) :: string
    character(kind=C_CHAR), dimension(len(string)+1) :: array
    integer :: i
    do i=1, len(string)
       array(i)=string(i:i)
    end do
    array(len(string)+1)=C_NULL_CHAR
  end function CString
  function quit()
    use general_routines
    logical :: quit
    quit = .false.
    if (kbhit() /= 0) then
       quit = (upcase(char(getch())) == 'Q')
    end if
  end function quit
  subroutine initwindow2(width,height)
    integer(C_INT), intent(in), value :: width,height
    call initwindow(width,height,CString('Windows BGI'),0,0)
  end subroutine initwindow2
  subroutine initwindow3(width,height,title)
    integer(C_INT), intent(in), value :: width,height
    character(C_CHAR), intent(in) :: title(*)
    call initwindow(width,height,title,0,0)
  end subroutine initwindow3
  subroutine initwindow4(width,height,title,left)
    integer(C_INT), intent(in), value :: width,height
    character(C_CHAR), intent(in) :: title(*)
    integer(C_INT), intent(in), value :: left
    call initwindow(width,height,title,left,0)
  end subroutine initwindow4
  subroutine initwindow5(width,height,title,left,top)
    integer(C_INT), intent(in), value :: width,height
    character(C_CHAR), intent(in) :: title(*)
    integer(C_INT), intent(in), value :: left,top
    call initwindow(width,height,title,left,top)
  end subroutine initwindow5
end module f03bgi
```

```
!
! Fortran Interface to the WinBGIm-3.6 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.
!
!
! Created     : Sep 16, 2010
! Last change : May 14, 2015
!

module gks_bgi
  use f03bgi
  implicit none
  integer, parameter, private :: MAX_NT = 10
  integer, private :: key,clipping = CLIP_ON,ivp_image(0:MAX_NT,4)
  logical, private :: graphics_on = .false.
  type (viewporttype), private :: display
  real(DP), private :: wn(0:MAX_NT,4) = 0.0_DP,vp(0:MAX_NT,4) = 0.0_DP,&
       coeff(0:MAX_NT,4)  ! sx,sy,tx,ty
  real(DP), private :: wk_wn(4) = 0.0_DP,wk_vp(4) = 0.0_DP,wkvp_eff(4) = 0.0_DP
  real(DP), private :: xmax,ymax,xxmax,yymax,&
       sx,sy,tx,ty
  private :: true_wkvp,get_transformation,transform,get_intersection,&
       setup_transformation,build_transformations,setup,&
       drawpoly2,drawpoly3,fillpoly2,fillpoly3,&
       initgraphics0,initgraphics1,initgraphics1b,initgraphics2,&
       initgraphics2b,initgraphics3,initgraphics4,initgraphics5,&
       outtext1,outtext3,&
       s2x,s2y,x2s,y2s
  interface gks_init
     module procedure initgraphics0,initgraphics1,initgraphics1b,&
          initgraphics2,initgraphics2b,initgraphics3,initgraphics4,&
          initgraphics5
  end interface gks_init
  interface gks_polyline
     module procedure drawpoly2,drawpoly3
  end interface gks_polyline
  interface gks_fillpoly
     module procedure fillpoly2,fillpoly3
  end interface gks_fillpoly
  interface gks_text
     module procedure outtext1,outtext3
  end interface gks_text
contains
  ! NT or WKT c(:) = (Sx,Sy,Tx,Ty)
  subroutine get_transformation(w,v,c)
    real(DP), intent(in) :: w(:),v(:)
    real(DP), intent(out) :: c(:)
    c(1) = (v(2)-v(1))/(w(2)-w(1))
    c(2) = (v(4)-v(3))/(w(4)-w(3))
    c(3) = v(1)-c(1)*w(1)
    c(4) = v(3)-c(2)*w(3)
  end subroutine get_transformation
  ! NT or WKT c(:) = (Sx,Sy,Tx,Ty)
  subroutine transform(u,c,v)
    real(DP), intent(in) :: u(:),c(:)
    real(DP), intent(out) :: v(:)
    v(1) = c(3)+c(1)*u(1)
    v(2) = c(3)+c(1)*u(2)
    v(3) = c(4)+c(2)*u(3)
    v(4) = c(4)+c(2)*u(4)
  end subroutine transform
  subroutine get_intersection(p,q)
    real(DP), intent(in) :: p(:)
    real(DP), intent(inout) :: q(:)
    integer, parameter :: L = 1,R = 2,B = 3,T = 4
    real(DP) :: m(4)  ! aux
    ! Default to the effective wk vp
    m = p
    ! Verifying the intersection
    if ((q(L) <= p(R)).and.(q(R) >= p(L))&
         .and.(q(B) <= p(T)).and.(q(T) >= p(B))) then
       m(L) = max(p(L),q(L))
       m(R) = min(p(R),q(R))
```

```
      m(B) = max(p(B),q(B))
      m(T) = min(p(T),q(T))
    end if
    q = m
  end subroutine get_intersection
  ! Computing sx,sy,tx,ty for k−th WC to Screen transformation
  subroutine setup_transformation(k)
    integer, intent(in) :: k
    real(DP) :: vp_image(4),vpis(4),a(4),c(4)  ! c(:) = (sxx,txx,syy,tyy)
    if (k < 0.and.MAX_NT < k) then
       write(*,*) 'Error from setup_transformation():'
       write(*,*) 'K =',k,' out of range!'
       call closegraph()
       stop
    end if
    ! Computing the coefficients of WKT (s_xi,s_eta,t_xi,t_eta)
    call get_transformation(wk_wn,wkvp_eff,c)
    ! Now computing the 'image' of vp
    call transform(vp(k,:),c,vp_image)
    call get_intersection(wkvp_eff,vp_image)
    ! Now we have to transform vp_image to real screen coordinates, ivp_image
    a(4) = ymax−1.0_DP
    a(1) = (xmax−1.0_DP)/xxmax
    a(2) = −a(4)/yymax   ! Ys increases toward bottom
    a(3) = 0.0_DP
    call transform(vp_image,a,vpis)
    ! ivp_image(:) = (left,right,bottom,top)
    ivp_image(k,:) = nint(vpis)
    ! Finally, we can start to compute our WC to Screen transformation
    ! coefficients after resetting the viewport (stored as fp values)
    vpis(2) = ivp_image(k,2)−ivp_image(k,1)
    vpis(3) = ivp_image(k,3)−ivp_image(k,4)
    vpis(1) = 0
    vpis(4) = 0
    ! First : Computing the NT coefficients (su,sv,tu,tv)...
    call get_transformation(wn(k,:),vp(k,:),a)
    !
    ! ...then the WC to DC transornation coefficients:
    !
    !   a(3) = T_xi  = t_xi+s_xi*tu
    !   a(4) = T_eta = t_eta+s_eta*tv
    !   a(1) = S_xi  = s_xi*su
    !   a(2) = S_eta = s_eta*sv
    !
    a(3) = c(3)+c(1)*a(3)
    a(4) = c(4)+c(2)*a(4)
    a(1) = c(1)*a(1)
    a(2) = c(2)*a(2)
    ! Now the DC to Screen coefficients (Sx,Sy,Tx,Ty)...
    call get_transformation(vp_image,vpis,c)
    !
    ! ...then the WC to Screen transformation coefficients:
    !
    !   coeff(k,1) = sx = Sx*S_xi
    !   coeff(k,2) = sy = Sy*S_eta
    !   coeff(k,3) = tx = Tx+Sx*T_xi
    !   coeff(k,4) = ty = Ty+Sy*T_eta
    !
    coeff(k,1) = c(1)*a(1)
    coeff(k,2) = c(2)*a(2)
    coeff(k,3) = c(3)+c(1)*a(3)
    coeff(k,4) = c(4)+c(2)*a(4)
  end subroutine setup_transformation
  ! Computes the effective wk viewport, vp_eff, so that it has the same
  ! aspect ratio as wk window
  subroutine true_wkvp(w,v,v_eff)
    real(DP), intent(in) :: w(:),v(:)
    real(DP), intent(out) :: v_eff(:)
    real(DP) :: alpha,beta,xx,yy
    v_eff = v
    ! Computing aspect ratios Y/X
    xx = w(2)−w(1)
    yy = w(4)−w(3)
    alpha = yy/xx
    xx = v(2)−v(1)
    yy = v(4)−v(3)
    beta = yy/xx
    if (beta < alpha) then
```

```
      xx = yy/alpha
      v_eff(2) = v_eff(1)+xx
   else
      yy = xx*alpha
      v_eff(4) = v_eff(3)+yy
   end if
end subroutine true_wkvp
subroutine build_transformations()
   integer :: k
   ! Getting the true wk vp, wkvp_eff
   call true_wkvp(wk_wn,wk_vp,wkvp_eff)
   do k = 0,MAX_NT
      call setup_transformation(k)
   end do
end subroutine build_transformations
subroutine setup()
   integer :: i
   if (.not.graphics_on) then
      graphics_on = .true.
      call getviewsettings(display)
      clipping = CLIP_ON
      ! Screen dimensions in pixels
      xmax = display%right−display%left+1.0_DP
      ymax = display%bottom−display%top+1.0_DP
      ! Normalization (in [0,1]) of screen dimensions
      yymax = max(xmax,ymax)
      xxmax = xmax/yymax
      yymax = ymax/yymax
      ! Default NT (0): cannot be modified!
      wn(0,:) = (/ 0.0_DP,1.0_DP,0.0_DP,1.0_DP /)
      vp(0,:) = (/ 0.0_DP,1.0_DP,0.0_DP,1.0_DP /)
      ! Now the other NT
      do i = 1,MAX_NT
         wn(i,:) = wn(0,:)
         vp(i,:) = vp(0,:)
      end do
      ! Default WKT: the vp is defaulted to the full display (normalized!)
      wk_wn = (/ 0.0_DP,1.0_DP,0.0_DP,1.0_DP /)
      wk_vp = (/ 0.0_DP,xxmax,0.0_DP,yymax /)
      call build_transformations()
      ! Init to default: sx,sy,tx,ty
      call gks_selnt(0)
   else
      write(*,*) 'Error from setup():'
      write(*,*) 'The graphics is already enabled!'
      call closegraph()
      stop
   end if
end subroutine setup
subroutine drawpoly2(numpoints,points)
   integer, intent(in) :: numpoints
   real(DP), intent(in) :: points(2*numpoints)
   integer :: k,ke,ko,ipoints(2*numpoints)
   do k = 1,numpoints
      ke = k+k
      ko = ke−1
      ipoints(ko) = x2s(points(ko))
      ipoints(ke) = y2s(points(ke))
   enddo
   call drawpoly(numpoints,ipoints)
end subroutine drawpoly2
subroutine drawpoly3(n,x,y)
   integer, intent(in) :: n
   real(DP), intent(in) :: x(:),y(:)
   integer :: k,ke,ko,ipoints(2*n)
   do k = 1,n
      ke = k+k
      ko = ke−1
      ipoints(ko) = x2s(x(k))
      ipoints(ke) = y2s(y(k))
   enddo
   call drawpoly(n,ipoints)
end subroutine drawpoly3
subroutine fillpoly2(numpoints,points)
   integer, intent(in) :: numpoints
   real(DP), intent(in) :: points(2*numpoints)
   integer :: k,ke,ko,ipoints(2*numpoints)
   do k = 1,numpoints
```

```
      ke = k+k
      ko = ke-1
      ipoints(ko) = x2s(points(ko))
      ipoints(ke) = y2s(points(ke))
    enddo
    call fillpoly(numpoints,ipoints)
  end subroutine fillpoly2
  subroutine fillpoly3(n,x,y)
    integer, intent(in) :: n
    real(DP), intent(in) :: x(:),y(:)
    integer :: k,ke,ko,ipoints(2*n)
    do k = 1,n
      ke = k+k
      ko = ke-1
      ipoints(ko) = x2s(x(k))
      ipoints(ke) = y2s(y(k))
    enddo
    call fillpoly(n,ipoints)
  end subroutine fillpoly3
  subroutine initgraphics0()
    integer :: gdriver = DETECT,gmode,errorcode
    call initgraph(gdriver,gmode,CString(''))
    ! ================================
    !  Read result of initialization
    ! ================================
    errorcode = graphresult()
    ! ====================
    !  An error occurred
    ! ====================
    if (errorcode /= grOk) then
      write(*,*) 'Graphics error: ',grapherrormsg(errorcode)
      write(*,'(A)',advance='NO') 'Press any key to halt:'
      key = getch()
      write(*,*) key
      ! ===========
      !  Terminate
      ! ===========
      stop
    endif
    call setup()
  end subroutine initgraphics0
  subroutine initgraphics1(window_size)
    integer, intent(in) :: window_size
    call init_window(window_size,window_size)
    call setup()
  end subroutine initgraphics1
  subroutine initgraphics1b(title)
    character(len=*), intent(in) :: title
    integer, parameter :: WINDOW_SIZE = 600
    call init_window(WINDOW_SIZE,WINDOW_SIZE,CString(title))
    call setup()
  end subroutine initgraphics1b
  subroutine initgraphics2(window_xsize,window_ysize)
    integer, intent(in) :: window_xsize,window_ysize
    call init_window(window_xsize,window_ysize)
    call setup()
  end subroutine initgraphics2
  subroutine initgraphics2b(window_size,title)
    integer, intent(in) :: window_size
    character(len=*), intent(in) :: title
    call init_window(window_size,window_size,CString(title))
    call setup()
  end subroutine initgraphics2b
  subroutine initgraphics3(window_xsize,window_ysize,title)
    integer, intent(in) :: window_xsize,window_ysize
    character(len=*), intent(in) :: title
    call init_window(window_xsize,window_ysize,CString(title))
    call setup()
  end subroutine initgraphics3
  subroutine initgraphics4(window_xsize,window_ysize,title,left)
    integer, intent(in) :: window_xsize,window_ysize,left
    character(len=*), intent(in) :: title
    call init_window(window_xsize,window_ysize,CString(title),left)
    call setup()
  end subroutine initgraphics4
  subroutine initgraphics5(window_xsize,window_ysize,title,left,top)
    integer, intent(in) :: window_xsize,window_ysize,left,top
    character(len=*), intent(in) :: title
```

```
  call init_window(window_xsize,window_ysize,CString(title),left,top)
  call setup()
end subroutine initgraphics5
subroutine outtext1(text)
  character(len=*), intent(in) :: text
  call outtext(CString(text))
end subroutine outtext1
subroutine outtext3(x,y,text)
  real(DP), intent(in) :: x,y
  character(len=*), intent(in) :: text
  call outtextxy(x2s(x),y2s(y),CString(text))
end subroutine outtext3
function s2x(pixel_x)  ! The inverse
  real(DP) :: s2x
  integer, intent(in) :: pixel_x
  s2x = (pixel_x−tx)/sx
end function s2x
function s2y(pixel_y)  ! The inverse
  real(DP) :: s2y
  integer, intent(in) :: pixel_y
  s2y = (pixel_y−ty)/sy
end function s2y
function x2s(x)
  integer :: x2s
  real(DP), intent(in) :: x
  x2s = nint(tx+sx*x)
end function x2s
function y2s(y)
  integer :: y2s
  real(DP), intent(in) :: y
  y2s = nint(ty+sy*y)
end function y2s
subroutine gks_arc(x,y,stangle,endangle,r)
  real(DP), intent(in) :: x,y,stangle,endangle,r
  call ellipse(x2s(x),y2s(y),nint(stangle),nint(endangle),&
       abs(x2s(r)−x2s(0.0_DP)),abs(y2s(r)−y2s(0.0_DP)))
end subroutine gks_arc
subroutine gks_bar(x1,x2,y1,y2)
  real(DP), intent(in) :: x1,x2,y1,y2
  call bar(x2s(x1),y2s(y1),x2s(x2),y2s(y2))
end subroutine gks_bar
subroutine gks_bar3d(x1,x2,y1,y2,depth,itop_flag)
  real(DP), intent(in) :: x1,x2,y1,y2,depth
  integer, intent(in) :: itop_flag
  call bar3d(x2s(x1),y2s(y1),x2s(x2),y2s(y2),&
       abs(x2s(depth)−x2s(0.0_DP)),itop_flag)
end subroutine gks_bar3d
subroutine gks_box(x1,x2,y1,y2)
  real(DP), intent(in) :: x1,x2,y1,y2
  call rectangle(x2s(x1),y2s(y1),x2s(x2),y2s(y2))
end subroutine gks_box
subroutine gks_circle(x,y,r)
  real(DP), intent(in) :: x,y,r
  call ellipse(x2s(x),y2s(y),0,360,&
       abs(x2s(r)−x2s(0.0_DP)),abs(y2s(r)−y2s(0.0_DP)))
end subroutine gks_circle
subroutine gks_close()
  key = getch()
  call closegraph()
end subroutine gks_close
subroutine gks_dot(x,y,color)
  real(DP), intent(in) :: x,y
  integer, intent(in) :: color
  call putpixel(x2s(x),y2s(y),color)
end subroutine gks_dot
subroutine gks_ellipse(x,y,stangle,endangle,a,b)
  real(DP), intent(in) :: x,y,stangle,endangle,a,b
  call ellipse(x2s(x),y2s(y),nint(stangle),nint(endangle),&
       abs(x2s(a)−x2s(0.0_DP)),abs(y2s(b)−y2s(0.0_DP)))
end subroutine gks_ellipse
subroutine gks_fillellipse(x,y,a,b)
  real(DP), intent(in) :: x,y,a,b
  call fillellipse(x2s(x),y2s(y),&
       abs(x2s(a)−x2s(0.0_DP)),abs(y2s(b)−y2s(0.0_DP)))
end subroutine gks_fillellipse
subroutine gks_fillarea(x,y,border_color)
  real(DP), intent(in) :: x,y
  integer, intent(in) :: border_color
```

```
  call floodfill(x2s(x),y2s(y),border_color)
end subroutine gks_fillarea
subroutine gks_getimage(x1,x2,y1,y2,bitmap)
  real(DP), intent(in) :: x1,x2,y1,y2
  type (C_PTR), value :: bitmap
  call getimage(x2s(x1),y2s(y2),x2s(x2),y2s(y1),bitmap)
end subroutine gks_getimage
subroutine gks_getmouseclick(kind,x,y)
  integer, intent(in) :: kind
  real(DP), intent(out) :: x,y
  integer :: pixel_x,pixel_y  ! Location of the mouse click
  call getmouseclick(kind,pixel_x,pixel_y)
  x = s2x(pixel_x)
  y = s2y(pixel_y)
end subroutine gks_getmouseclick
function gks_getpixel(x,y) result(color)
  real(DP), intent(in) :: x,y
  integer :: color
  color = getpixel(x2s(x),y2s(y))
end function gks_getpixel
function gks_getx() result(x)
  real(DP) :: x
  x = s2x(getx())
end function gks_getx
function gks_gety() result(y)
  real(DP) :: y
  y = s2y(gety())
end function gks_gety
function gks_imagesize(x1,x2,y1,y2) result(sz)
  real(DP), intent(in) :: x1,x2,y1,y2
  integer :: sz
  sz = imagesize(x2s(x1),y2s(y2),x2s(x2),y2s(y1))
end function gks_imagesize
subroutine gks_line(x1,y1,x2,y2)
  real(DP), intent(in) :: x1,y1,x2,y2
  call line(x2s(x1),y2s(y1),x2s(x2),y2s(y2))
end subroutine gks_line
subroutine gks_linerel(dx,dy)
  real(DP), intent(in) :: dx,dy
  call linerel(x2s(dx)−x2s(0.0_DP),y2s(dy)−y2s(0.0_DP))
end subroutine gks_linerel
subroutine gks_lineto(x,y)
  real(DP), intent(in) :: x,y
  call lineto(x2s(x),y2s(y))
end subroutine gks_lineto
subroutine gks_moverel(dx,dy)
  real(DP), intent(in) :: dx,dy
  call moverel(x2s(dx)−x2s(0.0_DP),y2s(dy)−y2s(0.0_DP))
end subroutine gks_moverel
subroutine gks_moveto(x,y)
  real(DP), intent(in) :: x,y
  call moveto(x2s(x),y2s(y))
end subroutine gks_moveto
subroutine gks_pieslice(x,y,stangle,endangle,r)
  real(DP), intent(in) :: x,y,stangle,endangle,r
  call sector(x2s(x),y2s(y),nint(stangle),nint(endangle),&
       abs(x2s(r)−x2s(0.0_DP)),abs(y2s(r)−y2s(0.0_DP)))
end subroutine gks_pieslice
subroutine gks_putimage(x,y,bitmap,op)
  real(DP), intent(in) :: x,y
  type (C_PTR), value :: bitmap
  integer, intent(in) :: op
  call putimage(x2s(x),y2s(y),bitmap,op)
end subroutine gks_putimage
subroutine gks_sector(x,y,stangle,endangle,a,b)
  real(DP), intent(in) :: x,y,stangle,endangle,a,b
  call sector(x2s(x),y2s(y),nint(stangle),nint(endangle),&
       abs(x2s(a)−x2s(0.0_DP)),abs(y2s(b)−y2s(0.0_DP)))
end subroutine gks_sector
subroutine gks_selnt(k)
  integer, intent(in) :: k
  ! In absolute coordinate, so we don't need to open the full window
  ! viewport :−)
  call setviewport(ivp_image(k,1),ivp_image(k,4),&
       ivp_image(k,2),ivp_image(k,3),clipping)
  ! coeff(k,:) is computed elsewhere
  sx = coeff(k,1)
  sy = coeff(k,2)
```

```
    tx = coeff(k,3)
    ty = coeff(k,4)
  end subroutine gks_selnt
  subroutine gks_swn(k,x1,x2,y1,y2)
    integer, intent(in) :: k
    real(DP), intent(in) :: x1,x2,y1,y2
    if (k < 1.and.MAX_NT < k) then
       write(*,*) 'Error from gks_swn():'
       write(*,*) 'K =',k,' out of range!'
       call closegraph()
       stop
    end if
    wn(k,:) = (/ x1,x2,y1,y2 /)
    call setup_transformation(k)
  end subroutine gks_swn
  subroutine gks_svp(k,x1,x2,y1,y2)
    integer, intent(in) :: k
    real(DP), intent(in) :: x1,x2,y1,y2
    if (k < 1.and.MAX_NT < k) then
       write(*,*) 'Error from gks_svp():'
       write(*,*) 'K =',k,' out of range!'
       call closegraph()
       stop
    end if
    vp(k,:) = (/ x1,x2,y1,y2 /)
    call setup_transformation(k)
  end subroutine gks_svp
  subroutine gks_swkwn(x1,x2,y1,y2)
    real(DP), intent(in) :: x1,x2,y1,y2
    wk_wn = (/ x1,x2,y1,y2 /)
    call build_transformations()
  end subroutine gks_swkwn
  subroutine gks_swkvp(x1,x2,y1,y2)
    real(DP), intent(in) :: x1,x2,y1,y2
    wk_vp = (/ x1,x2,y1,y2 /)
    call build_transformations()
  end subroutine gks_swkvp
  subroutine gks_sclip(clip)
    integer, intent(in) :: clip
    clipping = clip
  end subroutine gks_sclip
  subroutine gks_schrsz(x,y)
    real(DP), intent(in) :: x,y
    type (textsettingstype) ::  textinfo
    integer mul_x,mul_y,div_x,div_y
    call gettextsettings(textinfo)
    call settextstyle(textinfo%font,textinfo%direction,1)
    ! Getting the current pixel dimensions
    div_x = textwidth(CString('H'))
    div_y = textheight(CString('H'))
    ! New pixel dimensions
    mul_x = nint(x*abs(x2s(1.0_DP)−x2s(0.0_DP)))
    mul_y = nint(y*abs(y2s(1.0_DP)−y2s(0.0_DP)))
    call setusercharsize(mul_x,div_x,mul_y,div_y)
  end subroutine gks_schrsz
end module gks_bgi
```

```
!
! Fortran Interface to the WinBGIm-3.6 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/MINGW64 shells)
!
!   cd ~/work/WinBGIm-3.6p
!   g++ -c -O3 winbgim.cxx
!   ar rcs libWinBGIm.a winbgim.o
!
!   mkdir -p ~/programming/lib/msys2
!   mv libWinBGIm.a ~/programming/lib/msys2
!   rm winbgim.o
!
!   cd demo
!   g++ -mwindows -I.. ball_cursor.cpp ../winbgim.cxx -o ball_cursor.out
!
!   cd ~/programming/WinBGIm-fortran/apps
!
!   rm -rf {*.mod,~/programming/modules/*} && \
!   gfortran -O3 -Wall -mwindows \
!     -J ~/programming/modules \
!     ~/programming/basic-modules/basic_mods.f90 \
!     ../{f03bgi.f90,gks_bgi.f90} cernlib_mods.f90 cobra_bgi.f90 \
!     -L ~/programming/lib/msys2 -lWinBGIm -lstdc++ \
!     -o cobra_bgi.out
!
!     ./cobra_bgi.out    ( or: ./cobra_bgi.out < cobra_bgi.dat )
!
! In MINGW64, add '-static' and:
!
!   msys2 ==> mingw64
!   cobra_bgi.out ==> cobra_bgi
!
!
! DESCRIPTION
!    Motion in the magnetic field of COBRA spectrometer.
!
! REFERENCES
!    Press  Numerical Recipes  C.U.P
!    Karlen D., Computational Physics (WEB notes)
!    Karlen D.,Computers in Science (WEB notes)
!    G95 Manual and web page
!    GSL Manual
!

module cobra_field
  use kind_consts, only: DP
  use math_consts
  use io_consts
  use elliptic_k_e
  implicit none
  integer :: nCoils
  integer :: ierr
  character(len=*), parameter :: FCOILS = 'ashield.coils'
  real(DP), parameter :: BF0 = 12.6_DP  ! BF0 in KG
  real(DP), dimension(:), allocatable :: zco,rco,ico
  real(DP) :: field_qradius = -1.0_DP
contains
  subroutine magnet_on()
    ! ====================================================================
    !  The routine reads the coils data (RCO, ZCO, ICO, being the currents
    !  ICO not normalized), computes the normalization factor CF and
    !  normalizes the currents so to have a field with value BF0 at origin
    ! ====================================================================
    real(DP) :: r,z,curr,a2,az2,cf,fac
    integer :: k
    write(STDERR,*) '...now reading ',trim(FCOILS)
    open(20,file=FCOILS,status='OLD',action='READ')
    read(20,*) nCoils
    allocate(zco(nCoils),STAT=ierr)
    if (ierr /= 0) call error('ZCO: Allocation request denied')
```

```
  allocate(rco(nCoils),STAT=ierr)
  if (ierr /= 0) call error('RCO: Allocation request denied')
  allocate(ico(nCoils),STAT=ierr)
  if (ierr /= 0) call error('ICO: Allocation request denied')
  ! =====================
  !  Now computing B0...
  ! =====================
  cf = 0.0_DP
  do k = 1,nCoils
     read(20,*) r,z,curr
     field_qradius = max(field_qradius,r)
     rco(k) = r
     zco(k) = z
     ico(k) = curr
     a2 = r**2
     az2 = a2+z**2
     cf = cf+curr*a2/(az2*sqrt(az2))
  enddo
  close(20)
  cf = BF0/(PI*cf)
  fac = 5.D3*cf
  write(STDERR,*) 'Magnetic field computed... B−FIELD'
  write(STDERR,*)
  write(STDERR,*) 'Requested field at (0,0), BF0 : ',BF0,' KG'
  write(STDERR,*) 'Nominal field with I, Bz(0,0) : ',BF0/fac,' KG'
  write(STDERR,*) 'BF0 is obtained with C*I,   C : ',fac
  write(STDERR,*) 'Field radial region,        R : ',field_qradius,' cm'
  write(STDERR,*)
  ! ================================================
  !  Normalization of currents. Note...vec = vec*scalar
  ! ================================================
  ico = ico*cf
  ! Now, we need only the sqaure
  field_qradius = field_qradius**2
end subroutine magnet_on
subroutine magnet_off()
  if (allocated(ico)) deallocate(ico,stat=ierr)
  if (ierr /= 0) call error('ICO: Deallocation request denied')
  if (allocated(rco)) deallocate(rco,stat=ierr)
  if (ierr /= 0) call error('RCO: Deallocation request denied')
  if (allocated(zco)) deallocate(zco,stat=ierr)
  if (ierr /= 0) call error('ZCO: Deallocation request denied')
end subroutine magnet_off
subroutine bcalc(r,z,br,bz)
  real(DP), intent(in) :: r,z
  real(DP), intent(out) :: br,bz
  ! ================================================================
  !  The routine computes the components BR and BZ of field at point
  !  (R,Z) (remember the field symmetry).
  !  DELIEC(), DELIKC() are new user entry name; old: DELLIE(), DELLIK
  ! ================================================================
  real(DP) :: zz,a2,cq,ck,ce,p,q
  integer :: k
  br = 0.0_DP
  bz = 0.0_DP
  do k = 1,nCoils
     ! ================================================
     !  If the point is near a coil, less then 1 mm,...
     ! ================================================
     if (abs(rco(k)−r) < 0.1_DP.and.abs(zco(k)−z) < 0.1_DP) then
        br = 0.0_DP
        bz = 0.0_DP
        return
     endif
     zz = z−zco(k)
     p = zz**2
     cq = (r+rco(k))**2+p
     ck = sqrt(4.0_DP*r*rco(k)/cq)
     cq = ico(k)/sqrt(cq)
     ce = (rco(k)−r)**2+p
     ce = deliec(ck)/ce
     ck = delikc(ck)
     p = p+r**2
     a2 = rco(k)**2
     q = a2−p
     p = p+a2
     bz = bz+cq*(ck+q*ce)
     if (r > 0.0_DP) br = br+cq*zz*(−ck+p*ce)
```

```
    enddo
    if (r > 0.0_DP) br = br/r
  end subroutine bcalc
  ! ======================================
  !  Computing the Magnetic Field: at X,Y,Z
  ! ======================================
  subroutine get_field(x,b)
    real(DP), intent(in) :: x(:)
    real(DP), intent(out) :: b(:)
    real(DP) :: r,z,br
    r = hypot(x(1),x(2))
    z = x(3)
    call bcalc(r,z,br,b(3))
    if (r > 0.0_DP) then
       br = br/r
       b(1) = br*x(1)
       b(2) = br*x(2)
    else
       b(1) = 0.0_DP
       b(2) = 0.0_DP
    endif
  end subroutine get_field
end module cobra_field

module solution
  use cobra_field
  use randoms
  use gks_bgi
  implicit none
  integer :: num_eve = 10
  real(DP) :: s0 = 0.0_DP,s1 = 200.0_DP, stp = 0.5_DP    ! s0,s1,stp in cm
  real(DP) :: c_x = 0.0_DP,c_y = 0.0_DP,delta_x = 50.0_DP,delta_y = 50.0_DP
  real(DP) :: x_min = 0.0_DP,x_max = 0.0_DP,y_min = 0.0_DP,y_max = 0.0_DP,&
       s_min = 0.0_DP,s_max = 0.0_DP
contains
  subroutine init_data()
    use get_data
    call get('Number of events, NUM_EVE = ',num_eve)
    call get('Track length, S0 = ',s0)
    call get('Track length, S1 = ',s1)
    call get('Track step, STP = ',stp)
    call get('Center X, C_X = ',c_x)
    call get('Center Y, C_Y = ',c_y)
    call get('Width X, DELTA_X = ',delta_x)
    call get('Width Y, DELTA_Y = ',delta_y)
    call init_rand()
  end subroutine init_data
  subroutine solve()
    use io_consts
    integer, parameter :: NEQ = 6,&
         WINDOWS_HEIGHT = 612,WINDOWS_WIDTH = 1200
    integer :: i,col = 1,max_colors
    real(DP) :: x1,x2,y1,y2,y0(NEQ) = 0.0_DP
    call gks_init(WINDOWS_WIDTH,WINDOWS_HEIGHT,'COBRA with BGI!')
    ! 0.51 = 612/1200
    call gks_swkwn(0.0_DP,1.0_DP,0.0_DP,0.51_DP)
    call gks_swkvp(0.0_DP,1.0_DP,0.0_DP,0.51_DP)
    ! We really need half...
    delta_x = delta_x/2
    delta_y = delta_y/2
    ! First view
    x1 = c_x-delta_x
    x2 = c_x+delta_x
    y1 = c_y-delta_y
    y2 = c_y+delta_y
    call gks_swn(1,x1,x2,y1,y2)
    call gks_svp(1,0.02_DP,0.49_DP,0.02_DP,0.49_DP)
    call gks_selnt(1)
    call setcolor(RED)
    call gks_box(x1,x2,y1,y2)
    ! Second view
    x1 = -180.0_DP
    x2 = -x1
    y1 = 30.0_DP
    y2 = 40.0_DP
    call gks_swn(2,x1,x2,y1,y2)
    call gks_svp(2,0.51_DP,0.98_DP,0.02_DP,0.49_DP)
    call gks_selnt(2)
```

```
   call setcolor(CYAN)
   call gks_box(x1,x2,y1,y2)
   max_colors = getmaxcolor()+1
   call magnet_on()
   ! Boundaries initialization
   x_min = 1E6_DP
   x_max = -x_min
   y_min = 1E6_DP
   y_max = -y_min
   s_min = s0
   s_max = s0
   write(STDERR,*)
   write(STDERR,'(A)',advance='NO') 'Processing...'
   do i = 1,num_eve
      col = mod(col+1,max_colors)
      if (col == 0) col = col+1
      call get_event_kinematics(y0)
      call event_tracking(col,s0,s1,y0)
      if (quit()) exit
   end do
   write(STDERR,'(A)') 'done!'
   call magnet_off()
   call gks_close()
end subroutine solve
subroutine get_event_kinematics(y0)
   real(DP), intent(out) :: y0(:)
   real(DP), parameter :: PHI1 = -60.0_DP*DEG2RAD,PHI2 = 60.0_DP*DEG2RAD,&
        CTHE1 = -0.35_DP,CTHE2 = 0.0_DP,&
        P = 52.8_DP
   real(DP) :: phi,cos_the,sin_the
   ! ======================================
   !  r = r1+(r2-r1)*rnd() ==> r in [r1,r2)
   ! ======================================
   phi = PHI1+(PHI2-PHI1)*get_rand()
   cos_the = CTHE1+(CTHE2-CTHE1)*get_rand()
   sin_the = sqrt(1.0_DP-cos_the**2)
   y0(1:3) = 0.0_DP
   y0(4:6) = P*(/ cos(phi)*sin_the,sin(phi)*sin_the,cos_the /)
end subroutine get_event_kinematics
subroutine event_tracking(col,s0,s1,y0)
   use runge_kutta
   integer, intent(in) :: col
   real(DP), intent(in) :: s0,s1,y0(:)
   integer :: num_equ
   real(DP) :: phi_max,rq,r_max,p(2),s,x(size(y0)),w(3*size(y0))
   num_equ = size(y0)
   ! ====================
   !  Initial conditions
   ! ====================
   x = y0
   s = s0
   p = x(1:2)
   rq = dot_product(p,p)
   ! Now, r_max is the square...
   r_max = rq
   call gks_selnt(1)
   call gks_dot(x(1),x(2),col)
   ! ====================
   !  Updating position
   ! ====================
   do while((kbhit() == 0).and.(s < s1))
      call drkstp(num_equ,stp,s,x,derivs,w)
      rq = dot_product(x(1:2),x(1:2))
      if (rq > r_max) then
         r_max = rq
         p = x(1:2)
      end if
      x_min = min(x(1),x_min)
      x_max = max(x(1),x_max)
      y_min = min(x(2),y_min)
      y_max = max(x(2),y_max)
      s_min = min(s,s_min)
      s_max = max(s,s_max)
      call gks_dot(x(1),x(2),col)
      if (rq > field_qradius) exit
   enddo
   ! phi in degrees
   phi_max = atan2(p(2),p(1))/DEG2RAD
```

```
    r_max = sqrt(r_max)
    call gks_selnt(2)
    call gks_dot(phi_max,r_max,WHITE)
  end subroutine event_tracking
  subroutine derivs(s,y,f)
    real(DP), intent(in) :: s,y(*)
    real(DP), intent(out) :: f(*)
    ! ================================================================
    !  Compute the derivatives of the equations to be integrated (with
    !  Runge-Kutta method).
    !  The equations of motion are
    !
    !     dR/dS = P/|P|
    !     dP/dS = C00*(P/|P|) X B
    !
    !  with
    !
    !     R(1:3)     the vector radius, in cm;
    !     P(1:3)     the momentum, in MeV/c;
    !     S = |V|*dt trajectory arc length, in cm;
    !     |P|        the momentum magnitude, in MeV/c;
    !     B(1:3)     the field, in KG;
    !     C00        = 0.299792458, positron charge in ((MeV/c)/cm)/KG;
    !     X          the cross product.
    !
    !     Y(1:3) = R(1:3) is the position of particle.
    !     Y(4:6) = P(1:3) is the momentum of particle.
    ! ================================================================
    real(DP), parameter :: C00 = 0.299792458_DP
    real(DP), save :: b(3),p
    ! ============================
    !  Momentum magnitude in MeV/c
    ! ============================
    p = sqrt(y(4)**2+y(5)**2+y(6)**2)
    call get_field(y(1:3),b)
    f(1:3) = y(4:6)/p
    f(4) = C00*(f(2)*b(3)-f(3)*b(2))
    f(5) = C00*(f(3)*b(1)-f(1)*b(3))
    f(6) = C00*(f(1)*b(2)-f(2)*b(1))
  end subroutine derivs
  subroutine print_data()
    delta_x = x_max-x_min
    delta_y = y_max-y_min
    c_x = x_min+0.5_DP*delta_x
    c_y = y_min+0.5_DP*delta_y
    write(*,*)
    write(*,*) 'EFFECTIVE VALUES:'
    write(*,*) '   C_X     = ',c_x
    write(*,*) '   C_Y     = ',c_y
    write(*,*) '   DELTA_X = ',delta_x
    write(*,*) '   DELTA_Y = ',delta_y
    write(*,*)
    write(*,*) 'S_MIN = ',s_min
    write(*,*) 'S_MAX = ',s_max
  end subroutine print_data
end module solution

program cobra_bgi
  use solution
  implicit none
  call init_data()
  call solve()
  call print_data()
end program cobra_bgi

subroutine error(chMsg)
  use io_consts
  implicit none
  character(len=*), intent(in) :: chMsg
  ! ========================================================
  !  Print the message chMsg and stop the program.
  !  This routine MUST be called with an unrecoverable error
  ! ========================================================
  write(STDERR,*)
  write(STDERR,*) 'Run-time error...'
  write(STDERR,*) chMsg
  write(STDERR,*) '...now exiting to system.'
  write(STDERR,*)
```

```
  stop
end subroutine error

! function hypot(x,y)
!   use kind_consts
!   implicit none
!   real(DP) :: hypot
!   real(DP), intent(in) :: x,y
!   real(DP) :: absx,absy
!   absx = abs(x)
!   absy = abs(y)
!   if (absx > absy) then
!      hypot = absx*sqrt(1.0_DP+(absy/absx)**2)
!   else
!      if (absy == 0.0_DP) then
!         hypot = 0.0_DP
!      else
!         hypot = absy*sqrt(1.0_DP+(absx/absy)**2)
!      end if
!   end if
! end function hypot
```