Inside GWindows

Introduction

GWindows is an object-oriented framework that simplifies programming applications for Microsoft Windows in Ada. The original author of GWindows is David Botton. Mr. Botton maintains a GWindows web page on his adapower web site (http://www.adapower.com/gwindows).

GWindows is released under the GNAT modified GNU Public License (GMGPL). A loose translation of the GMGPL is: you may link your own program with code covered by the GMGPL without redistributing your own code, but you may not sell GMGPL'd code as your own. Note: This interpretation is probably wrong, so read the header in one of the GWindows modules and do a little research on your own to find the correct meaning.

This book analyzes of the GWindows framework. Basic knowledge of programming Windows using the Win32 API is assumed. A good book on programming windows using the Win32 API is *Programming Windows, by Charles Petzold*.

The Windows Class

Windows SDK programs often register a separate class for each kind of window used in a Windows application. A class is registered by setting up a structure of type WNDCLASS and calling the Windows API function ***RegisterClass*** with the structure. The WNDCLASS structure is defined in the winuser.h header file of the Win32 SDK as:

```
typedef struct {
    UINT  style;
    WNDPROC  lpfnWndProc;
    int  cbClsExtra;
    int  cbWndExtra;
    HINSTANCE  hInstance;
    HICON  hIcon;
    HCURSOR  hCursor;
    HBRUSH  hbrBackground;
    LPCTSTR  lpszMenuName;
    LPCTSTR  lpszClassName;
} WNDCLASS, *PWNDCLASS;
```

The corresponding Ada structure is defined in the package spec gwindows.base as:

```
  type WNDCLASS is
      record
         style         : Interfaces.C.unsigned :=
           CS_HREDRAW or CS_VREDRAW or CS_DBLCLKS or CS_OWNDC;
         lpfnWndProc   : System.Address        := WndProc'Address;
         cbClsExtra    : Interfaces.C.int      := 0;
         cbWndExtra    : Interfaces.C.int      := 0;
         hInstance     : GWindows.Types.Handle := 0;
         hIcon         : GWindows.Types.Handle := 0;
         hCursor       : GWindows.Types.Handle := 0;
         hbrBackground : GWindows.Types.Handle :=
           Interfaces.C."-" (GWindows.Colors.COLOR_BTNFACE, 1);
         lpszMenuName  : Pointer_To_GChar_C    := null;
         lpszClassName : Pointer_To_GChar_C    := null;
      end record;
```

In GWindows all windows created use the same class. This class is registered during elaboration of the package body GWindows.base. The WNDCLASS definition includes default values for each field. The hInstance, hIcon, and lpszClassName fields are updated just before calling RegisterClass.

```
   Window_Class.hInstance := GWindows.Internal.Current_hInstance;
   Window_Class.hIcon := LoadIcon;
   Window_Class.lpszClassName := GWindows.Internal.Window_Class_Name
     (GWindows.Internal.Window_Class_Name'First)'Access;
   RegisterClass;
```

While the values in WNDCLASS are used to create all windows in GWindows, some values can be (and are) modified by GWindows after a window is created using this class, more on this later.

The hInstance value in WNDCLASS is the handle to the program instance. This value is obtained from the global variable Current_hInstance located in the GWindows.Internal package spec. The global variable is set up during elaboration of the GWindows.Internal package as the return value from the function rts_get_hInstance in the Ada runtime.

```
function hInst return Interfaces.C.long;
pragma Import (C, hInst, "rts_get_hInstance");
--  Load Instance information from Ada Windows run time
…
Current_hInstance := hInst;
```

The hIcon value in WNDCLASS is a handle to the icon associated with the window. The value is set by calling the Windows API function LoadIcon. The binding to LoadIcon is defined in the package body gwindows.base. In setting up the hIcon value in WNDCLASS, default values for selection of the icon are used.

```
function LoadIcon
  (hInstance  : GWindows.Types.Handle := 0;
   lpIconName : Integer := IDI_APPLICATION)
  return GWindows.Types.Handle;
pragma Import (StdCall, LoadIcon,
                "LoadIcon" & Character_Mode_Identifier);
```

The lpszClassName value in WNDCLASS is a pointer to a null terminated string. The string value is defined in the package spec GWindows.Internal as “GWindows_Class”.

```
Window_Class_Name : GString_C :=
  GWindows.GStrings.To_GString_C ("GWindows_Class");
--  Class name used for all GWindows windows
```

The GWindows module GStrings encapsulates the type of string used by GWindows. This package is described by the documentation included in the GWindows distribution.

The callback procedure for the window class is set to WndProc, located in the package body GWindows.Base.

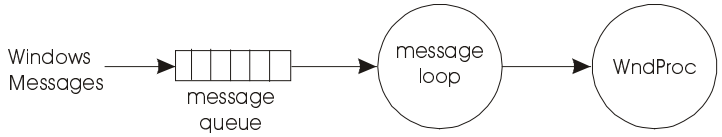
```
function WndProc
   (hwnd    : GWindows.Types.Handle;
    message : Interfaces.C.unsigned;
    wParam  : Interfaces.C.int;
    lParam  : Interfaces.C.int)
  return Interfaces.C.long
is
.
.
.
end WndProc;
```

There will be discussion on the implementation of WndProc later in this book.

Processing Windows Messages

The previous section explained GWindows uses the same class for all Windows. A natural question arises: if GWindows uses the same class for all windows, why don’t all windows in GWindows behave the same? The answer to this question is the subject of this section.

The following diagram illustrates messages processing under windows.



Windows sends most messages to programs through a message queue. Typical windows programs register a class using *RegisterClass*, create a window using *CreateWindow* or *CreateWindowEx*, show the window using *ShowWindow*, update the window using *UpdateWindow*, and then go into a message processing loop that repeated calls *GetMessage*, *TranslateMessage*, and *DispatchMessage* and terminate when GetMessage returns a null value. If the last rambling sentance made sense, you already know enough about windows to follow this document. If not ***run!***

When a program calls DispatchMessage windows does a little behind the scene bookkeeping and then calls the window procedure registered with the class for the window (*WndProc*). Windows also sends some messages directly to the window procedure bypassing the message queue.

The Windows SDK describes the window procedure:

```
LRESULT CALLBACK WindowProc(
  HWND hwnd,
  UINT uMsg,
  WPARAM wParam,
  LPARAM lParam );
```

Where:

```
hwnd:
  Handle to the window.
uMsg
  Specifies the message.
wParam
  Specifies additional message information. The contents of this parameter depend on the value of the
       uMsg parameter.
lParam
  Specifies additional message information. The contents of this parameter depend on the value of the
       uMsg parameter.
```

Programs written with the Windows SDK often implement the window procedure using a case statement on uMsg, taking different action depending on the message received. If a message is received that the program does not handle, the program calls a default windows message handler (DefWindowProc) and returns the result. If the program does process the message, a value of zero is returned indicating that the message has been processed.

In the GWindows window procedure, GWindows uses an object you create as a descendent of Base_Window_Type to invoke methods for each window message. Details of how GWindows obtains access to this object are described in the pages that follow.

The Windows SDK defines messages in the windows.h header file of the form WM_<message type> where <message type> gives the nature of the message. For example: WM_HSCROLL is used when a user clicks on a horizontal scroll bar.

The following elements of code are in the gwindows.base package spec:

```
type Scroll_Request_Type is ( End_Scroll,
                              First,
                              Last,
                              Previous_Unit,
                              Next_Unit,
                              Previous_Page,
                              Next_Page,
                              Thumb_Set,
                              Thumb_Drag);

type Scroll_Event is access
  procedure (Window  : in out GWindows.Base.Base_Window_Type'Class;
             Request : in     Scroll_Request_Type);

procedure On_Horizontal_Scroll_Handler
  (Window  : in out Base_Window_Type;
   Handler : in     Scroll_Event);

procedure Fire_On_Horizontal_Scroll
  (Window  : in out Base_Window_Type;
   Request : in     Scroll_Request_Type);

procedure On_Horizontal_Scroll
  (Window  : in out Base_Window_Type;
   Request : in     Scroll_Request_Type;
   Control : in     Pointer_To_Base_Window_Class);
```

```
type Base_Window_Type is
  new Ada.Finalization.Limited_Controlled with
   record
    …
    On_Horizontal_Scroll_Event : Scroll_Event          := null;
    …
```

The On_Horizontal_Scroll procedure is called by the window procedure when a WM_HSCROLL message is received. The window procedure decodes the parameters received by the window procedure giving the request and the control that made the request. The fragment of code that does this decoding is located in the gwindows.base package in the procedure WndProc. Most of the details of the WndProc are omitted here to focus on the WM_HSCROLL handling.

```
function WndProc
  (hwnd    : GWindows.Types.Handle;
   message : Interfaces.C.unsigned;
   wParam  : Interfaces.C.int;
   lParam  : Interfaces.C.int)
  return Interfaces.C.long
is
   ...
   WM_HSCROLL                : constant := 276;
   ...
   case message is
   ...
      when WM_HSCROLL =>
         declare
            Request : Scroll_Request_Type;
            Control      : Pointer_To_Base_Window_Class :=
              Window_From_Handle (GWindows.Types.Handle (lParam));
         begin
            case GWindows.Utilities.Low_Word (wParam) is
               when SB_LINELEFT =>
                  Request := Previous_Unit;
               when SB_LINERIGHT =>
                  Request := Next_Unit;
               when SB_PAGELEFT =>
                  Request := Previous_Page;
               when SB_PAGERIGHT =>
                  Request := Next_Page;
               when SB_THUMBPOSITION =>
                  Request := Thumb_Set;
               when SB_THUMBTRACK =>
                  Request := Thumb_Drag;
               when SB_LEFT =>
                  Request := First;
               when SB_RIGHT =>
                  Request := Last;
               when others =>
                  Request := End_Scroll;
            end case;

            On_Horizontal_Scroll (Win_Ptr.all,
                                  Request,
                                  Control);

            return 0;
         end;
   ...
```

Note that after On_Horizontal_Scroll is called a zero is returned by the message procedure signaling windows that the message has been handled. The default implementation of On_Horiziontal_Scroll is also located in the gwindows.base body:

```
procedure On_Horizontal_Scroll
  (Window  : in out Base_Window_Type;
   Request : in     Scroll_Request_Type;
   Control : in     Pointer_To_Base_Window_Class)
is
begin
   if Control /= null then
      On_Horizontal_Scroll (Control.all, Request, null);
   else
      Fire_On_Horizontal_Scroll (Window, Request);
   end if;
end On_Horizontal_Scroll;
```

If the source of the scroll message is a control window, the scroll message is redirected to that window by calling On_Horizontal_Scroll for that window. When On_Horizontal_Scroll is called with Control set to null, Fire_On_Horizontal_Scroll is called. The default implementation for Fire_On_Horizontal_Scroll is also located in the gwindows.base package.

```
procedure Fire_On_Horizontal_Scroll
  (Window  : in out Base_Window_Type;
   Request : in     Scroll_Request_Type)
is
begin
   if Window.On_Horizontal_Scroll_Event /= null then
      Window.On_Horizontal_Scroll_Event (Base_Window_Type'Class (Window),
                                         Request);
   end if;
end Fire_On_Horizontal_Scroll;
```

The Fire_On_Horizontal_Scroll procedure checks to see if the value On_Horizontal_Scroll_Event in the window object contains a valid pointer to a procedure for handling the message. If a valid pointer is present the procedure is called, otherwise the fire routine does nothing.

The On_Horizontal_Scroll_Handler procedure assigns the value passed in as Handler to the On_Horizontal_Scroll_Event field in the window object.

```
procedure On_Horizontal_Scroll_Handler
   (Window  : in out Base_Window_Type;
    Handler : in     Scroll_Event)
is
begin
   Window.On_Horizontal_Scroll_Event := Handler;
end On_Horizontal_Scroll_Handler;
```

This technique of dispatching messages is used throughout GWindows. If you wish to change the default handling of a message (which is often to do nothing), either derive a new window object that overrides the On_XXX method, or create a stand alone procedure and make that procedure the handler for a message using the On_XXX_Handler procedure.

How the Window Object is Associated with a Window

GWindows creates an object derived from Base_Window_Type for each window created. The type Base_Window_Type is defined as a private type derived from Ada.Finalization.Limited in the package gwindows.base (file: gwindows-base.ads).

```
type Base_Window_Type is
   new Ada.Finalization.Limited_Controlled with
    record
       HWND             : GWindows.Types.Handle        := 0;
       ParentWindowProc : Windproc_Access              := null;
       haccel           : GWindows.Types.Handle        := 0;
       MDI_Client       : Base_Window_Access           := null;
       Keyboard_Support : Boolean                      := False;
       Is_Control       : Boolean                      := False;
       Last_Focused     : GWindows.Types.Handle        := 0;
       Is_Dynamic       : Boolean                      := False;
       Modal_Result     : Integer                      := 0;
       In_Dialog        : Boolean                      := False;
       Dock             : Dock_Type                    := None;
       --  Event Handlers
       On_Create_Event            : Action_Event          := null;
       On_Pre_Create_Event        : Pre_Create_Event      := null;
       On_Destroy_Event           : Action_Event          := null;
       On_Context_Menu_Event      : Location_Action_Event := null;
       On_Horizontal_Scroll_Event : Scroll_Event          := null;
       On_Vertical_Scroll_Event   : Scroll_Event          := null;
    end record;
```

A class wide pointer to an object derived from Base_Window_Type is associated with each window created. This pointer is stored in a window *property* by the procedure Set_GWindow_Object located in the gwindows.base package body.

```
procedure Set_GWindow_Object (HWND   : GWindows.Types.Handle;
                              Object : Pointer_To_Base_Window_Class)
is
   procedure SetProp
     (handle   : GWindows.Types.Handle        := HWND;
      lpString : Interfaces.C.unsigned_short  :=
        GWindows.Internal.GWindows_Object_Property_Atom;
      hData    : Pointer_To_Base_Window_Class := Object);
   pragma Import (StdCall, SetProp, "SetProp" & Character_Mode_Identifier);
begin
   SetProp;
end Set_GWindow_Object;
```

GWindows includes several types of windows all derived from Base_Window_Type. A window of type Window_Type is defined in the gwindows.windows package spec. The procedure *Create* for Window_Type is also declared in the gwindows.windows package spec and of course implemented in the gwindows.windows pacakge body.

```
procedure Create
  (Window     : in out Window_Type;
   Title      : in     GString     := "";
   Left       : in     Integer     := GWindows.Constants.Use_Default;
   Top        : in     Integer     := GWindows.Constants.Use_Default;
   Width      : in     Integer     := GWindows.Constants.Use_Default;
   Height     : in     Integer     := GWindows.Constants.Use_Default;
   Is_Dynamic : in     Boolean     := False;
   CClass     : in     GString     := "")
is
   C_Title  : GString_C := GWindows.GStrings.To_GString_C (Title);
   Win_HWND : GWindows.Types.Handle;
   Style    : Interfaces.C.unsigned := WS_OVERLAPPEDWINDOW;
   ExStyle  : Interfaces.C.unsigned := 0;
   CClass_C : GString_C := GWindows.GStrings.To_GString_C (CClass);
begin
   On_Pre_Create (Window_Type'Class (Window), Style, ExStyle);

   if CClass = "" then
      Win_HWND := CreateWindowEx (lpWindowName => C_Title,
                                  dwExStyle    => ExStyle,
                                  dwStyle      => Style,
                                  x            => Left,
                                  y            => Top,
                                  nWidth       => Width,
                                  nHeight      => Height);
   else
      Win_HWND := CreateWindowEx (lpWindowName => C_Title,
                                  lpClassName  => CClass_C,
                                  dwExStyle    => ExStyle,
                                  dwStyle      => Style,
                                  x            => Left,
                                  y            => Top,
                                  nWidth       => Width,
                                  nHeight      => Height);
   end if;

   GWindows.Base.Link (Window, Win_HWND, Is_Dynamic);
   On_Create (Window_Type'Class (Window));
end Create;
```

This default Create procedure of Window_Type sets up a number of local variables in the correct form to call CreateWindowEx. The On_Pre_Create procedure is called to allow the style or extended style to be overridden. To specialize On_Pre_Create either derive a new type from Window_Type and override the On_Pre_Create procedure or register a handler using On_Pre_Create_Handler.

After calling On_Pre_Create the Create procedure creates the window using the Win32 API call CreateWindowEx. The Ada binding to the Win32 API call defines default values for many of the parameters.

```
function CreateWindowEx
  (dwExStyle    : Interfaces.C.unsigned;
   lpClassName  : GString_C :=
     GWindows.Internal.Window_Class_Name;
   lpWindowName : GString_C;
   dwStyle      : Interfaces.C.unsigned;
   x            : Integer;
   y            : Integer;
   nWidth       : Integer;
   nHeight      : Integer;
   hwndParent   : GWindows.Types.Handle   := 0;
   hMenu        : GWindows.Types.Handle   := 0;
   hInst        : GWindows.Types.Handle   :=
     GWindows.Internal.Current_hInstance;
   lpParam      : Interfaces.C.long       := 0)
  return GWindows.Types.Handle;
pragma Import (StdCall, CreateWindowEx,
               "CreateWindowEx" & Character_Mode_Identifier);
```

What follows is work in progress…

Numerous methods are defined in gwindows.base that may be overridden by types derived from Base_Window_Type. In the package gwindows.windows (file:gwindows-windows.ads) type Window_Type is derived from Base_Window_Type:

```
type Window_Type is new GWindows.Base.Base_Window_Type with private;
```

And adds private fields specific to Window_Type:

```
type Window_Type is new GWindows.Base.Base_Window_Type with
   record
      On_Focus_Event          : GWindows.Base.Action_Event   := null;
      On_Lost_Focus_Event     : GWindows.Base.Action_Event   := null;
      On_Size_Event           : Size_Event                   := null;
      On_Move_Event           : Location_Event               := null;
      On_Show_Event           : GWindows.Base.Action_Event   := null;
      On_Hide_Event           : GWindows.Base.Action_Event   := null;
      On_Close_Event          : Close_Event                  := null;
      On_MDI_Activate_Event   : GWindows.Base.Action_Event   := null;
      On_MDI_Deactivate_Event : GWindows.Base.Action_Event   := null;
      On_File_Drop_Event      : File_Drop_Event              := null;

      On_Mouse_Move_Event                      : Mouse_Event := null;
      On_Left_Mouse_Button_Down_Event          : Mouse_Event := null;
      On_Left_Mouse_Button_Up_Event            : Mouse_Event := null;
      On_Left_Mouse_Button_Double_Click_Event  : Mouse_Event := null;
      On_Right_Mouse_Button_Down_Event         : Mouse_Event := null;
      On_Right_Mouse_Button_Up_Event           : Mouse_Event := null;
      On_Right_Mouse_Button_Double_Click_Event : Mouse_Event := null;
      On_Middle_Mouse_Button_Down_Event        : Mouse_Event := null;
      On_Middle_Mouse_Button_Up_Event          : Mouse_Event := null;
      On_Middle_Mouse_Button_Double_Click_Event : Mouse_Event := null;

      On_Character_Up_Event   : Character_Event := null;
      On_Character_Down_Event : Character_Event := null;

      On_Menu_Hover_Event         : Hover_Event  := null;
      On_Menu_Select_Event        : Select_Event := null;
      On_Accelerator_Select_Event : Select_Event := null;

      On_Paint_Event            : Paint_Event                 := null;
      On_Erase_Background_Event : Paint_Event                 := null;
      On_Change_Cursor_Event    : GWindows.Base.Action_Event  := null;

      All_Keys : Boolean := False;

      Font_Handle : GWindows.Types.Handle := 0;
   end record;
```