JanusW

Version 1.30 © 1994 Peter Sawatzki
Date: 03/04/94 Buchenhof 3, D 58091 Hagen, Germany
CompuServe: 100031,3002

The JanusW.Zip is a collection of units for Borland Pascal for Windows. The zip file is called JanusW(n) because of a feature in the object tDialogWindow from the unit DialogWn: it let's you decide at runtime whether or not to display the dialog as a BorDlg or as a standard dialog; 'Janus' because of the roman God Janus with two faces.

JanusW contains these units:

DialogWn	the main unit: contains object tDialogWindow that enables the use of dialogs as MDI child windows
DynLink	contains object tDll to **dynamically** load and unload DLLs. Two instances of tDll dBWCC and dCtl3d contain definitions for Borland and Microsoft custom controls.
VBX	this unit enables support for VBX controls
Debug	useful functions for debugging output

Included are the following sample programs:

DlgTest	full featured test program to test tDialogWindows as MDI childs. This application may be used as a non-MDI application as well.
MinMdi	this is a minimal 10 line demo program to demonstrate the use of dialogs as MDI childs

See also:

Tips	tips for the usage of tDialogWindow
New	whats new in this JanusW release
DEW	how to modify tEntryScreen object from TurboPowers Data Entry Workshop to coexist with tDialogWindow

Contributing:

People	these people contributed to the development of JanusW

the following people contributed to the development of JanusW:
Dan O. Butler (dob) [72134,633]
Andy Cook [71331,501]
Ron Loewy (rl) [100274,162]
Jeroen W. Pluimers (jwp)
Max Stempfhuber (ms) [100140,2034]
Dean Wyant [75110,3253]

DialogWn

The unit DialogWn consists of three related objects: tDialogWindow, tAdvMdiWindow and tAdvApplication. While only tDialogWindow is necessary to make non-MDI dialogs, for MDI applications the use of tAdvMdiWindow and tAdvApplication is absolutely necessary to keep track of the dialog focus.

tDialogWindow

tDialogWindow is a descendant of tWindow that behaves like a modeless or (system-) modal   dialog. It may be used as a replacement for Borlands tDialog and tDlgWindow. Unlike the tDialog descendant tDlgWindow it inherits all features from tWindow making it easy to use tDialogWindow as a MDI child. Futhermore, tDialogWindow uses DnyLink to dynamically link to BWCC and Ctl3D. Any tDialogWindow instance can be forced to display itself as a normal, BWCC, or Ctl3D dialog (of course, the appropriate DLL must be available). For information about the usage of tDialogWindow see how to use tDialogWindow. See also: details of the tDialogWindow implementation

tAdvMdiWindow

This is a descendant of tMdiWindow that must be used to properly restore the focus of the MDI childs and to enable the KBHandlerWnd.

tAdvApplication

This is a descendant of tApplication that changes the application message loop for MDI dialog childreen to enable keyboard accelerators. Furthermore, it helps keep track of the dialog focus.
For an example of a minimal program that implements dialogs as MDI childs see the demo program MinMdi

DynLink

DynLink is a unit that takes care of dynamically linking Windows DLLs. It is used by DialogWn to load the Borland Custom Control library BWCC or the Microsoft CTL3D library.

Debug

A unit to route WriteLn()s to the debugging terminal or the DbWin window

BWCC

BWCC is Borlands Custom Control library to give your dialogs the Borland chiseled steel look

Ctl3D

Ctl3D is a Custom Control Library by Microsoft to give your dialogs a cool 3D look

BIVBX10

BIVBX10 is a .DLL file from the Borland C++ 4.0 compiler package. In JanusW it is used to provide access to Visual Basic Custom Controls (VBX)

DlgTest

DlgTest demonstrates the use of tDialogWindows as modeless and modal dialog windows and MDI child windows. It also demonstrates the effects of forcing dialogs to normal, BWCC or Ctl3D appearance.
creates a user customizable combination of a StdDlg, BorDlg and/or MS Ctl3D dialog

- demonstrates the use of "non-standard" MDI child styles
- enables Microsoft Ctl3D look for all type of dialogs

uses BWCC.DLL and Ctl3D if present.

MinMdi

MinMdi is a very simple program that shows how easy it is to use dialogs as MDI child windows with JanusW. For an enhanced example see DlgTest.Pas

How to use tDialogWindow

[to be written]

Details of the tDialogWindow implementation

[to be written]

Tips on using JanusW

Classnames

tDialogWindow receives information about its class directly from the resource: unlike Borlands tDlgWindow that must additionally get the classname from the OWL GetClassName method, the tDialogWindow object is smart enaugh to read the classname from the dialog resource and return it as a result of a GetClassName call. So you should **never** override GetClassName for tDialogWindow descendants: instead specify a unique classname in the properties dialog box of Resource Workshop only. Notice: using a **non-unique** classname leads to MDI child windows with inactive colors and other nice effects!

Making a modeless dialog or an MDI child dialog window

to make a modeless dialog window or to insert a dialog as an MDI child window you use Application^.MakeWindow:

```
   Application^.MakeWindow(New(pDialogWindow,Init(@Self, 'MyDialog')));
```

Making a modal dialog

To make a modal dialog with tDialogWindow, use *ExecDialogWindow* as you would use ExecDialog with the standard OWL tDialog object:

```
   aDialog:= New(pDialogWindow, Init(@Self, 'MyDialog'));
   If ExecDialogWindow(aDialog)=Id_Ok Then ....
```

Do not dispose the dialog after the ExecDialogWindow call, ExecDialogWindow does this for you as ExecDialog does it for tDialog objects!

Dialog styles

Specify style WS_POPUP for your dialogs. Although this style is not allowed for MDI child windows, tDialogWindow converts this window style to WS_CHILD if used under MDI. This makes it easy to use your dialogs as MDI childs and as popup dialogs if needed

MDI child windows with non-standard style

If you want to create non-standard MDI child windows e.g. child windows with non-sizeable frames, windows with no system menu etc., you must do the following (see DlgTest.Pas for an example):

1. override the InitClient method like this:

```
         Procedure aMDIWindow.InitClientWindow;
         Begin
         ClientWnd:= New(pMdiClient, Init(@Self));
         With ClientWnd^.Attr do
                 Style:= Style Or MdiS_AllChildStyles
         End;
```

2. specify **all** attributes for your child windows in Resource Workshop! MDI now uses no default style. This applies especially to the style WS_VISIBLE! If you do not specify this style your MDI child windows are created invisible and you have to use ShowWindow() to make 'em visible.

Scrollbars for the MDI Client window

If you like to have scrollbars for your MDI client window, modify InitClientWindow like this:

```
   Procedure aMDIWindow.InitClientWindow;
   Begin
      ClientWnd:= New(pMdiClient, Init(@Self));
      With ClientWnd^.Attr do
      Style:= StyleOr ws_VScroll Or ws_HScroll {Or MdiS_AllChildStyles}
   End;
```

Dynamic custom control support

tDialogWindow automatically and **dynamically** loads the following custom control libraries on demand:

1. BWCCbwcc.DLL, if class 'BorDlgxxxx' is specified as the dialog class or DlgStyle is set to ForceBor
2. Ctl3D.DLLctl3d, if the bit 'EnableCtl3D' is set in the DlgStyle

3. BIVBX10.DLLbivbx10 is loaded if the dialog resource contains a control of the VBcontrol class
Notice that none of the above libraries are loaded for dialogs that do not use one of the custom control.

Data Entry Workshop

it is possible to use the DialogWn unit together with TurboPowers Data Entry Workshop. For a demonstration of tDialogWindows as ancestors of their tEntryScreen, $define "DEW" and recompile the DlgTest program.

Versions used/supported by JanusW

1. the DialogWn unit is written for Borland Pascal 7.0x and does not work without modifications together with TPW 1.5.
2. the DialogWn unit is and will not be tested with Windows 3.0
3. the DialogWn unit is tested with Ctl3D version > 2.01.

Whats new in this JanusW release

Versions of JanusW prior to 1.25 called the standard Windows DefDlgProc() function to support subtile Windows quirks like closing an opened ComboListBox when the user starts to drag a dialog. Beginning with version 1.26 tDialogWindow does all this by itself. This means, it does not register the DlgExtra class bytes necessary to be DefDlgProc() compatible. *Tip*: **Do not call DefDlgProc() in tDialogWindow descendants!**

Version 1.27 adds support for VBX controls via Borlands BIVBX10.DLL

tDialogWindow and TurboPowers Data Entry Workshop

The tEntryScreen object from TurboPowers DEW (Data Entry Workshop) descends from the standard OWL object tDlgWindow. This topic explains how to change the source of DEW to make tEntryScreen descend from tDialogWindow. After applying the changes, the tEntryScreen object may be used as a MDI child window like any other tDialogWindow descendant.

The following three changes are necessary to OODEWCC.PAS:

1. add a 'Uses DialogWn'
2. make tEntryScreen descend from tDialogWindow instead of tDlgWindow (of course <g>)
3. disable the GetWindowClass and GetClassName methods: tDialogWindow retrieves the dialog class name directly from the resource and does it's own BorDlg/non-BorDlg/Ctl3D modifications ('Janus' property)

Here are the changes to the source of OODEWCC.PAS:

```
Uses
   ...
{$IfDef Janus}
   DialogWn,
{$EndIf}
{$IfDef Janus}
   tEntryScreenParent = tDialogWindow;
{$Else}
   tEntryScreenParent = tDlgWindow;
{$EndIf}
   PEntryScreen = ^TEntryScreen;
TEntryScreen = object(tEntryScreenParent) {Object corresponding to an entry screen}
...
{$IfNDef Janus}
   procedure GetWindowClass(var AWndClass : TWndClass); virtual;
   function  GetClassName : PChar; virtual;
{$EndIf}
...
end;
constructor TEntryScreen.Init(AParent : PWindowsObject; ATitle : PChar; var Buffer);
{-Initialize a TEntryScreen object}
begin
   tEntryScreenParent.Init(AParent, ATitle);
   ...
end;
{$IfNDef Janus}
... code for GetWindowClass and GetClassName
{$EndIf}
```

The unit VBX

VBX is a unit that takes care of dynamically linking to Borlands BIVBX10.DLL. It is used by DialogWn to support use of VBX controls in tDialogWindow.

The VBX unit contains several objects to implement the access to VBX controls:

dVbx	the dVbx variable is an instance of tVbx. It is used to link to BIVBX10.DLL and provide basic access to Vbx controls.
tVbxControl	defines an OWL wrapper object for a VBX control.

For users who dont want to use the famous tDialogWindow object for their dialogs:

tVbxDialog	a tDialog descendant that may contain VBX controls
tVbxDlgWindow	a tDldWindow descendant that may contain VBX controls

See also:

VbxInfo	a program to generate Pascal units from VBX files
VbxDemo	a demo program that uses VBX controls
DlgTest	another demo of Vbx controls and the usage of Vbx controls in MDI child windows

Important:

Important	this is necessary to use VBX controls in your programs

dVbx object

The dVbx variable is an instance of object tVbx and provides access to all documented functions exported by BIVBX10.DLL. tVbx is a descendant of tDll, an object for dynamic link library support. As with all tDll descendants you may call dVbxs methods without any pre-initialization. For example to create a instance of a certain Vbx control - say BIGAUGE - on the fly, all you have to do is the following:

```
Ctl:= dVbx.VBXCreate(hWindow, 100, gauge.vbx, bigauge, test, 0, 1, 1, 100, 100, 0);
```

Because dVbx has not loaded BIVBX10.DLL before, it first loads BIVBX10.DLL, calls dVbx.VbxInit, then loads gauge.vbx and finally creates the bigauge control.

Most of the time one will be using the methods from an instance of tVbxControl object. Consequently here is only a documentation of some of the functions exported from tVbx:

VBXGetHctl: Function (window: hWnd): hCtl;
This function returns the VBX control handle associated with the window *window* or Nil if *window* is not a valid VBX control

VBXGetHwnd: Function (control: hCtl): hWnd;
This function returns the window handle associated with the VBX control *control* or 0 if *control* is not a valid VBX control.

VBXCreate: Function (windowParent: hWnd; id: Integer;
lib, cls, title: pChar; style: LongInt;
x, y, w, h: Integer; aFile: hFormFile): hCtl;
This function creates a new instance of the control *cls* located in the VBX library *lib*.
The *style* argument specifies the control window style and can be set to 0 to use the default style.
The *file* argument specifies a form file and should be set to 0 for dynamically created controls. This function returns Nil if it is unable to load the VBX library and create the control.
x, *y*, *w*, and *h* are related system coordinates.

VBXInitDialog: Function (window: hWnd; instance: tHandle; id: pChar): Bool;
This function is used to initialize a dialog window *window* loaded from a resource *id* (located in *instance*) by creating VBX controls for each child window of class VBControl located in the dialog template. It should be called by the tWindow/tDialog/tDialogWindow/tDlgWindow object in the SetupWindow method. It returns TRUE if successful, or FALSE if an error occurs. Resource *id* must be of rt_DlgInit type.

tVbxControl object

Fields

Ctl (hCtl)
a 32 bit handle associated with every Visual Basic Custom Control

VbxName (pChar)
Name of the VBX library passed from an Init Constructor

VbxClass (pChar)
Class of VBX control passed from an Init Constructor

InitData (tHandle)
handle to default property data eventually passed from an Init Constructor

Class Methods

```
Constructor Init     (aParent: pWindowsObject; anId: Integer;
                     aVbxName, aVbxClass, aTitle: pChar;
                     x, y, w, h: Integer;
                     Len: LongInt; Data: Pointer);
```
Constructs a VBX control of class *aVbxClass* located in VBX library *aVbxName* with title *aTitle*.

Constructor InitResource(aParent: pWindowsObject; anId: Integer);
Associates already created VBX control *anId* with an object instance of tVbxControl.

Destructor Done; Virtual;

Object Methods

Function Create: Boolean; Virtual;

Procedure wmVbxFireEvent (Var Msg: tMessage); Virtual wm_First+wm_VbxFireEvent;
Msg.lParam is a pointer to a record of type tVbxEvent. This record contains the field *EventIndex* and determines what kind of event is fired from a VBX control. The wmVbxFireEvent method tries to find if a method exists in the VMT table for this event and routes the event to the appropriate method. If no method for the event exists, the event is routed to the DefaultEventProc.

Procedure DefaultEventProc (Var Event: tVbxEvent); Virtual;
This procedure is called if no special event handler exists for a certain event.

Function GetHCtl: hCtl;
return the VBX control handle associated with the tVbxControl instance.

Object Methods for Properties

Function GetNumProps: Integer;

Function GetPropIndex (Name: pChar): Integer;

Function GetPropName (Index: Integer): pChar;

Function GetPropType (Index: Integer): Integer;

Function IsArrayProp (Index: Integer): Bool;

Function GetProp (Index: Integer; Var Value): Bool;

Function GetPropByName (Name: pChar; Var Value): Bool;

Function GetStrProp (Index: Integer; Dst: pChar): Bool;

Function SetProp (Index: Integer; Value: LongInt): Bool;

Function SetPropByName (Name: pChar; Value: LongInt): Bool;

Object Methods for Events

Function GetNumEvents: Integer;

Function GetEventIndex (Name: pChar): Integer;

Function GetEventName (Index: Integer): pChar;

Object Methods for Vbx Methods

Function Method (aMethod: Integer; Var Args): Bool;

Function AddItem (Index: Integer; Const Item: pChar): Bool;

Function Drag (Action: Integer): Bool;

Function Move (x, y, w, h: LongInt): Bool;

Function Refresh: Bool;

Function RemoveItem (Index: Integer): Bool;

VbxInfo program

The VbxInfo program creates a wrapper unit from a VBX library.

The VbxDemo program

VbxDemo is a simple demo that shows how to use Vbx controls in tDialogWindow descendants. It subclasses a Vbx control via InitResource.

tVbxDialog and tVbxDlgWindow objects

tVbxDialog and tVbxDlgWindow are tDialog and tDlgWindow descendants that enable the usage of Vbx controls for standard OWL dialogs. This is done by linking to dVbx object before the dialog is created and by calling dVbx.VbxInitDialog in SetupWindow. Controls of the class VbControl are now converted to corresponding Vbx controls by VbxInitDialog.

Furthermore both objects reroute the wm_VbxFireEvent message to object instances of tVbxControl if the controls are initialized via tVbxControltVbxCtl.InitResource

VBX control jump vector

Every program that uses VBX controls must have a jump vector at address SS:20h installed: it is absolutely neccessary to add the following definition as the **first** typed constant statement to your main program to safely use VBX controls:

```
Const
   VbxValidation: tVbxValidation = cVbxValidation;
```

And to ensure that BP7's smartlinker does not remove this info, one must 'use' the ValidationInfo at least once via a RegisterVBX(VbxValidation) call.
The above makes sure that BIVBX10.DLL does not overwrite data at DS:20h when the VBX jump vector gets installed. Consequently the dVBX object refuses to link to BIVBX10 if the above conditions are not met!