Indigo Software

VCL Widget Pack Volume 1

The Indigo Software VCL Widget Pack Volume 1.0 contains the 14 Delphi VCL components along with their source code. The programmer is free to use these components royalty free in their own applications. The programmer may not, however distribute any source included on this disk, even if said source has been modified by the programmer, unless express permission is obtained from Indigo Software.

Support:

Any questions, comments, or suggestions regarding these components should be directed to:

Zane Rathwick
Indigo Software

Internet: ZaneR@aol.com
CIS: 74633,1314
WWW: http://users.aol.com/indigosoft/vcl

Wherever possible, both 16 and 32-bit versions of the code have been supplied, for users of Delphi 1.0 and 2.0. In

some cases, the changes in the operating system precluded providing a 32-bit version.

Installing The Components

Two sets of components are included. The 16-bit versions for Delphi 1.0 are in a self-extracting file called **INSTAL16.EXE**, and the 32-bit versions for Delphi 2.0 are in a self extracting file called **INSTAL32.EXE**. Run the correct executable for your installation.

You will be prompted for a location to extract the files to. Set this location to the directory in which you your other Delphi add-ons, such as C:\DELPHI\ADDONS. Once the files are extracted, start Delphi (if it is not already running), and install the component **IW_REG.PAS**. A new tab will appear on your tool palette called Indigo Widgets, and the new components will be installed there.

Included Components:

TCards
TDataComm
TDiamondPad
TDigitalLabel
TDigitalMarquee
TDragDrop
TFlyout
TGraphicPanel
TImageAspect
TMeter95
TMinMax
TMenuButton
TRollBar
TSysMenu

Below is a detailed listing of each component, its properties, events and methods.

TCards

Delphi 1.0 and 2.0

This VCL allows the user to easily create card games with Delphi.

PROPERTIES

CardBack: This is the style of the design used to draw the card when CardStyle is set to cdBack.
Possible values:

cbDiag	**cbHatch**	**cbRobot**
cbRoses	**cbGreenPlant**	**cbBluePlant**
cbLightFish	**cbDarkFish**	**cbShell**
cbCastle	**cbBeach**	**cbHand**

CardStyle: This depicts how the card will be drawn. Possible values are:

cdBack:	Draw card back design
cdFront:	Draw card front
cdO:	Draw placeholder card
cdX:	Draw invalid placeholder card

FaceValue: This is the face value of the card (ie 1-13)

Selected: When selected is true, card is drawn inversed, else card is drawn normal.

Suit: Depicts what suit the given card is. Possible values are:
suClubs
suDiamonds
suHearts
suSpades

Value: Depicts the value of the card using the formula:

```
Value:=FaceValue + (ord(Suit) * 13);
```

Conversely:

```
FaceValue:=Value - (ord(Suit) * 13);
```

IsRed: Boolean: True if card is red, false otherwise

IsFaceCard: Boolean: True if card value is > 10, false otherwise

USING TCARDS

To use this control, simply place it on your form, and edit the CardBack, CardStyle, Suit, and Value properties.

Sample Procedures:

```
procedure initdeck;
var
    i:integer;
begin
    {Initializes deck array, do once per program, or to reset}
    {deck[1..52] is a global array}
    for i:= 1 to 52 do
        deck[i]:=i;
end;


procedure shuffledeck;
var
    i,j,k,temp:integer;
begin
    {deck[1..52] is a global array}
    Randomize;
    For i := 1 To 10 do
        For j := 1 To 52 do
        begin
            k := trunc(1 + (52 * Random));
            temp := deck[j];
            deck[j] := deck[k];
            deck[k] := temp;
        end;
end;
```

TDataComm

Delphi 1.0

This VCL allows you to easily pass data between two applications running on the same computer without resorting to DDE. Use it to share data, find out if another instance of your application is already running, or even to remotely control one application from another.

PROPERTIES

Channel: This is the channel you will be broadcasting your data over. It is really a Windows message. The default is WM_USER + 99, but I sugggest you change it so as not to conflict with other programs that may be using this control. The value must be greater than WM_USER.

Data: This is a longint, and represents the data to be sent. This doesn't mean that you can only send 4 bytes of information, just typecast a pointer to your data as a long. For example:

```
Data := LongInt(pointer(MyBitmap));
```

When sending a pointer to an object, make sure that the object won't be destroyed before the reciever has a chance to read the data. The best way to do this is to only send pointers to global variables.

InData: This is the longint received by the control. Using the above example, to decode the data do the following:

```
Image1.picture.assign(TBitmap(pointer(InData)));
```

Note the use of Assign. Always make a local copy of the data as soon as possible to reduce the chance of the data being destroyed before you are finished.

METHODS

SendData: This is what actually triggers the data being sent.

EVENTS

OnReceiveData: This event is fired when data is received over the specified channel. At this point, InData will contain the LongInt received.

USING TDataComm

To use this control, simply place it on your form, and set the Channel property to some unique value over WM_USER.

Suppose you want to send a structure containing names and addresses from one application to another (both applications must contain a TDataComm control set to the same channel).

First create the structure:

```
Type
```

```
   TAddBook = Record
   Name : String;
   Address1 : String;
   Address2 : String;
   City : String;
   State : String;
   ZIP : String;
   Phone : String;
   Age : Integer;
   Other : LongInt;
end;

Var
   MyAddress : TAddBook;
```

Next, send the Data:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
   MyAddress.Name := 'Bob Jones';
   MyAddress.Address1 := '1234 Main Street';
   MyAddress.Address2 := 'Apt 4';
   {...fill in the rest of the structure}
   DataComm1.Data:=longint(pointer(MyAddress));
   DataComm1.SendData;
end;
```

In the receiving application, you should have the same structure as above defined. Then, add the following code to the OnReceiveData event:

```
procedure TForm1.DataComm1OnReceiveData(Sender: TObject);
var
   AnAddress : TAddBook;
begin
   AnAddress := TAddBook(pointer(InData));
   Label1.caption := AnAddress.Name;
   {..., etc.}
end;
```

TDigMarquee and TDigLabel

Delphi 1.0 and 2.0

This VCL gives you a multi-color scrolling marquee with digital letters and flashing effects as well as a static digital label with multiple colors.

TDIGMARQUEE

PROPERTIES

Caption: This is the text that will scroll across the control. A subset of the alphabet is supported, including all capital letters, numbers, and keyboard punctuation except ` and ~.

Formatting commands are embedded into the caption to alter text colors and to toggle flashing text. All commands are preceeded by a ~ character and are terminated with a space. Multiple commands may be stacked after a single ~. Available commands are:

~1 : (green text)
~2 : (red text)
~3 : (yellow text)
~4 : (cyan text)
~5 : (white text)
~f : (toggle flashing text)

Examples:

```
just plain green text
~2 red text~1 , green text
~5f white flashing text~4f , cyan non-flashing text
```

Enabled: This turns the timer on/off. If it is disabled, no text is visible

FlashTicks: This is the number of timer ticks between flashes of flashing text. This is NOT milliseconds. If the interval is set for 50, and FlashTicks is set for 5 then the letters will flash approx every every 250 milliseconds (5*50). If set to 0 then no flashing occurs. This only affects text flagged with ~f .

Interval: This is the interval between timer ticks. The smoothness of the scrolling text is a combination of the Interval and the Speed setting.

Loops: This is the number of times the text will scroll across the control before it stops. After it reaches this number, the OnFinishScrolling event will be fired to allow you to reset the caption or Loops, or other variables. This way, you can update the information like a stock ticker or a clock, or offer random tips, hints, or quotes. A negative number will scroll indefinately

Speed: This is the number of pixels to the left the text will move each timer tick. The smoothness of the scrolling text is a combination of the Interval and the Speed setting.

EVENTS

OnFinishScrolling:

This event is fired after the text has scrolled by the number of times specified by Loops. If Loops is negative, this event will never be called.

Example:

```
procedure TForm1.DigMarquee1FinishScrolling(Sender: TObject);
begin
   DigMarquee1.caption:='The time is: ~5f '+ timetostr(time);
   DigMarquee1.loops:=1;
end;
```

TDIGLABEL

PROPERTIES

Caption: This is the text that will scroll across the control. A subset of the alphabet is supported, including all capital letters, numbers, and keyboard punctuation except ` and ~.

Formatting commands are embedded into the caption to alter text colors and to toggle flashing text. All commands are preceeded by a ~ character and are terminated with a space. Multiple commands may be stacked after a single ~. Available commands are:

~1 : (green text)
~2 : (red text)
~3 : (yellow text)
~4 : (cyan text)
~5 : (white text)
~f : (toggle flashing text)

Examples:

```
just plain green text
~2 red text~1 , green text
```

TDiamondPad

Delphi 1.0 and 2.0

This VCL a 4-way spinner type control.

PROPERTIES

Delay: If the Interval property is set, the OnFire event will fire every Interval# of milliseconds, as long as the mouse is pressed down on the control. The Delay sets a delay in milliseconds before that starts to happen.

Interval: If the Interval property is set, the OnFire event will fire every Interval# of milliseconds, as long as the mouse is pressed down on the control.

Size: Two sizes of DiamondPads are available:
dsSmall (32x32)
dsLarge (48x48)

EVENTS

OnFire: The OnFire event will be triggered immediately upon the user clicking the mouse on the control, and, if the Interval property is set, every Interval# of milliseconds, starting after a delay of Delay milliseconds. When this event is called, you will be passed an ArrowPressed value which will be one of the following:
bpTop
bpRight
bpBottom
bpLeft.

Sample Event Procedure:

```
procedure TForm1.DiamondPad1Fire(Sender: TObject; ArrowPressed: TPressed);
begin
     case ArrowPressed of
     bpTop:    {Add code here};
     bpBottom: {Add code here};
     bpLeft:   {Add code here};
     bpRight:  {Add code here};
     end;
end;
```

TDragDrop

Delphi 1.0

This VCL allows you to make any windowed VCL a drag and drop target. It also parses all dropped files into a convenient
string list.

PROPERTIES

Component: This is the component that you want to accept dropped files.

FileList: This is a TStringList that will contain the name of all the dropped files.

EVENTS

ONFileDragDrop:

This is the event that is fired when files are dropped on the control specified by the Component property.

USING TDragDrop

To use TDragDrop, simply drop it on a form, and set the Component property to another control on the form (ie ListBox1). Then in the OnFileDragDrop event, code to do what you will with the dropped files.

For Example:

```
procedure TForm1.DragDrop1FileDragDrop(Sender: TObject);
var
   i:integer;
begin
   for i:=0 to DragDrop1.FileList.count-1 do
      listbox1.items.add(DragDrop1.FileList[i]);
end;
```

TFlyout

Delphi 1.0 and 2.0

This VCL gives you a flyout button similar to those found in image edition programs like Corel Draw or PhotoShop. A set of
flyout buttons acts similar to a two-dimentional array of radio buttons. Like radio buttons, only one flyout button within a
group may be selected, however, when a flyout button is clicked, it expands to offer you more choices.

PROPERTIES

ArrowColor: To show that the button offers more choices, the flyout button draws an arrow in the lower right-hand corner. This property allows you to select what color this arrow is drawn in.

Note that if NumGlyphs is set to 1 then the arrow won't be visible.

AutoFlyout: This property tells the flyout button to display its flyout every time the button is clicked, regardless of whether the value of that button is being changed or not. This is useful for novice users who might be unaware of how a flyout button works. By default, this property is set to false, and the user must click on a flyout button a second time to display the flyout.

Checked: Similar to a radio button or checkbox, this property tells whether or not a given button is selected.

Glyphs: This is the bitmap that is used to create the flyout buttons. This is similar to the glyph property of a Speedbutton in that multiple images (one for each button on the flyout) are placed on a single bitmap. Each image must be the same size. The flyout button will divide the bitmap evenly by the number in the NumGlyphs property to create the flyout buttons. Use the included sample glyphs as a template.

Index: This is a zero-based number that tells which flyout button is currently selected.

NumGlyphs: This is the number of images that are on the current Glyph, and also the number of buttons that are on the flyout. The flyout button will try to guess this number when you load a new glyph, based upon dividing the width by the height, so if your images are square, you don't need to set this property.

USING TFLYOUT

You use the flyout button like a radio button, except that it has an extra index property. For example:

```
If Flyout1.checked then
   case flyout1.index of
   0: {do something};
   1: {do something else};
   end
else if Flyout2.checked then
   {... repeat as necessary}
end;
```

or you might code in the OnClick event handler like this:

```
procedure TForm1.Flyout1Click(Sender: TObject);
begin
   case flyout1.index of
```

```
   0: {do something};
   1: {do something else};
   end;
end;
```

TGraphicPanel

Delphi 1.0 and 2.0

The TGraphicPanel component is a specialized image control that displays panel effects without the overhead of the Panel component. Useful when you want to visually group components, but don't need to physically group them. The Graphic Panel has an optional Win-95 look, as well as seperate background and foreground colors.

PROPERTIES

BackColor: The BackColor property defines the color that will be used to draw sunken surfaces on the panel.

BevelInner: A Graphic Panel component has two bevels, an outer bevel drawn next to the border of the control, and an inner bevel drawn inside the outer bevel. The width of the inner bevel is specified in the BevelWidth property in pixels. The BevelInner property determines the style of the inner bevel of a Graphic Panel component.

These are the possible values:

bvNone: No inner bevel exists.
bvLowered: The inner bevel is lowered.
bvRaised: The inner bevel is raised.

BevelOuter: A Graphic Panel component has two bevels, an outer bevel drawn next to the border of the control, and an inner bevel drawn inside the outer bevel. The width of the inner bevel is specified in the BevelWidth property in pixels. The BevelOuter property determines the style of the outer bevel of a Graphic Panel component.

These are the possible values:

bvNone: No outer bevel exists.
bvLowered: The outer bevel is lowered.
bvRaised: The outer bevel is raised.

BevelWidth: The BevelWidth property determines the width in pixels between the inner and the outer bevels of a Graphic Panel.

Bitmap: This property specifies a bitmap that will be tiled on the background of the panel. If this property is specified, then the Color and BackColor properties will be ignored.

Color: The Color property defines the color that will be used to draw raised surfaces on the panel.

Outline: The Outline property of the Graphic Panel control is a boolean value that toggles the appearance of the panel between the standard Windows 3.1 look and that of Win 95.

TextShadowColor: This is the color used to draw the drop shadow of the caption when TextStyle is set to tsDropShadow.

TextShadowWidth: This is the distance in pixels between the caption and the drop shadow of the caption when TextStyle is set to tsDropShadow.

TextStyle: This is the style used to draw the caption.

These are the possible values:

tsNormal: Flat text.
tsRaised: Raised text.
tsSunken: Etched text.
tsDropShadow: The text has a drop-shadow.

TImageAspect

Delphi 1.0 and 2.0

The TImageAspect control is similar to the TImage control except that it sizes a picture in the correct aspect ratio, regardless of the size, instead of stretching the image to fit both horizontally and vertically.

PROPERTIES

AutoSize: If autosize is true, the TImageAspect control will resize to the original image's dimensions, otherwise, the image will be resized accordingly.

TMenuButton

Delphi 1.0 and 2.0

This VCL gives you a button that displays a popup menu when pressed. The button looks similar to a SpeedButton, with an optional arrow to show the direction of the popup menu.

PROPERTIES

ArrowColor: To show that the button differs from a normal button, an optional arrow is drawn on the right side. This property allows you to select what color this arrow is drawn in.

Menu: This property tells the button what popup menu you want displayed when the button is depressed.

MenuAlignment:
maBottom: This causes the menu to popup aligned to the bottom-left corner of the button.
maRight: This causes the menu to popup aligned to the top-right corner of the button.

ShowArrow: This is a boolean that determines whether or not the arrow is drawn on the button.

TMeter95

Delphi 1.0

This VCL gives you a progress meter similar to the one used in Win95.

PROPERTIES

BackColor: This is the color used to fill the background of the control.

Ctl3D: This is a boolean value that toggles the 3D effects.

ForeColor: This is the color used to draw the progress portion of the control.

Max: This is the maximum value of the meter control, similar to the maximum value of a scrollbar. It must be greater than zero.

Min: This is the minimum value of the meter control, similar to the minimum value of a scrollbar. It must be zero or greater.

Value: This is the value (between Min and Max) of the meter control, similar the position property of a scrollbar.

TMinMax

Delphi 1.0 and 2.0

This VCL allows the user to easily set the minimum and maximum tracking sizes of their forms by intercepting the WM_GETMINMAXINFO message.

The WM_GETMINMAXINFO message is sent to a window whenever Windows needs the maximized position or dimensions of the window or needs the maximum or minimum tracking size of the window. The maximized size of a window is the size of the window when its borders are fully extended. The maximum tracking size of a window is the largest window size that can be achieved by using the borders to size the window. The minimum tracking size of a window is the smallest window size that can be achieved by using the borders to size the window.

PROPERTIES

MaxWidth: This is the maximum width you want your form to be able to be sized to, even while maximized. A setting of 0 will use the default size to be used.

MaxHeight: This is the maximum height you want your form to be able to be sized to, even while maximized. A setting of 0 will cause the default size to be used.

MinWidth: This is the minimum width you want your form to be able to be sized to. A setting of 0 will cause the default size to be used.

MinHeight: This is the minimum height you want your form to be able to be sized to. A setting of 0 will cause the default size to be used.

USING TMinMax

To use this control, simply place it on your form, and edit the MaxWidth, MaxHeight, MinWidth, and MinHeight

properties to whatever values you wish.

TRollBar

Delphi 1.0 and 2.0

The TRollBar Component gives your forms a mini title bar with optional roll-up capabilities. Simply drop it on your form and run your program to have a fully functioning mini title bar, or set some of the properties to change the defaults.

PROPERTIES

AllowRollup: The AllowRollup property determines whether or not the form will roll up/down when the rollup button is clicked.

AnimateRollup: The AnimateRollup property toggles the animation effect when the roll up/down button is clicked. The AnimateSteps property must be greater than 1 for this to take effect.

AnimateSteps: The AnimateSteps property determines the smothness and speed of the animation effect of the form rolling up/down. The smoother the animation, the slower the effect. The speed is based on the formula:

```
Speed:=Form.ClientHeight div AnimateSteps;
```

The form will shrink/grow in increments of Speed until it is fully resized.

IsRolledUp: The IsRolledUp property depicts whether or not the form is currently rolled up. Run-Time only.

MaxHeight: The MaxHeight property defines the maximum height a form will size to, even while maximized.

MaxWidth: The MaxWidth property defines the maximum width a form will size to, even while maximized.

MinHeight: The MinHeight property defines the minimum height a form will size to.

MinWidth: The MinWidth property defines the minimum width a form will size to.

PopupMenu: The PopupMenu property sets the menu that you want to use for the control menu. This menu (if set) will automatically popup when the control box is clicked.

ShowClose: The ShowClose property toggles whether or not the TRollBar displays the close form button.

Warning: Setting this property to false removes the user's ability to close this form unless some other mechanism (system menu or button) is provided.

True: The button is shown, and the user will be able to close the form by clicking on this button.

False: The button is not shown, and no form closing mechanism is provided.

ShowIcon: The ShowIcon property toggles whether or not the TRollBar displays the application's icon on its left edge (the system menu).

True: The icon is shown, and the system menu (if set in the PopupMenu property) is available

False: The icon is not shown, and no system menu is available.

ShowMinimize: The ShowRollup property toggles whether or not the TRollBar displays the minimize button.

True: The button is shown, and the user will be able to minimize the form by clicking on this button.

False: The button is not shown, and no minimization capabilities are available.

ShowRollup: The ShowRollup property toggles whether or not the TRollBar displays the roll-up/roll-down button.

True: The button is shown, and the user will be able to roll-up the form to just the size of the title bar, or roll it back down to its previous size.

False: The button is not shown, and no roll-up/roll-down capabilities are available.

EVENTS:

OnBeforeRollup: This event is fired immediately before the form is to be rolled up/down. Code here if you are using the GetMinMax message or something similar to limit the size of your form (ie MinHeight:=0).

OnAfterRollup: This event is fired immediately after the form is rolled up/down. Code here if you are using the GetMinMax message or something similar to limit the size of your form (ie MinHeight:=0).

TSysMenu

Delphi 1.0 and 2.0

This VCL allows the user to easily add menu items to the system menu of their programs. In addition, upon selecting
one of the added menu items, an event will be fired, passing a string containing the text of the menu item selected.

PROPERTIES

InsertBefore: This is the position on the system menu at which you want the new menu items placed. A value of -1 places the new items at the end

MenuItems: This is a TStringList containing the menu items you wish to add to the system menu. A "-" character denotes a seperator bar.

StartID: Each menu item must be assigned a unique ID#. The StartID denotes the number to use for the first menu item; each item thereafter is incremented by one.

EVENTS

OnFireMenu: This event is fired when one of the added menu items has been selected. A string will be passed to this event in the variable MenuCaption that contains the text of the menu item selected.

USING TSysMenu

To use this control, simply place it on your form, and edit the MenuItems property to contain the menu items you wish to add.

Example:

```
-
&About...
&Help
```

In the above example, the hyphen will be a seperator bar, followed by two menu items. Now change the InsertBefore property to 5. This will insert the new menu before the 5th menu entry in the system menu (this menu is zero-based, so the first entry is zero, and so on).

Now your menu should look like this:

```
Restore
Move
Size
Minimize
Maximize
-----------
About...
Help
-----------
Close
```

Double-click on the SysMenu control to edit its OnFireMenu event and alter its code as follows:

```
procedure TForm1.SysMenu1FireMenu(Sender: TObject; MenuCaption: String);
begin
     If MenuCaption = '&About...' then
          {code here to handle this event...}
     else If MenuCaption = '&Help' then
          {code here to handle this event...};
end;
```

SUPPORT:

Send comments and questions to:

Zane Rathwick
Indigo Software

CIS:	74633,1314
AOL:	ZaneR
Internet:	ZaneR@aol.com
U.S. Mail:	Zane Rathwick Indigo Software 4240 Park Newport Suite 308 Newport Beach, CA 92660

REGISTERING INDIGO WIDGET PACK

To register the Indigo Widget Pack and receive source code, send $50 (+$2 S&H) to:

Indigo Software
4240 Park Newport # 308
Newport Beach, CA 92660

Compuserve users may register on-line by going to keyword **GO SWREG** and ordering product number **10850**.

You can save the $2 shipping fee by specifying an internet address to have your control sent to instead of by mail.