Creating Visual Basic VBX's: The Basic Control

by Fred C. Hill

The first in a series of articles on the secrets of writing VBX controls in Borland Pascal

Everyone from the top down will tell you that to *really* program in Windows you must use C or C++. With the release of TPW 1.0 we've all learned different, but in a lot of ways those people are correct. The tool makers have listened and with few exceptions the tools needed to do a lot of things *easy* in Windows are all written for C and C++. What a lot of developers don't realize is that with a little ingenuity a lot of those C tools can be made to work with Borland Pascal.

From the early days of Windows 3.0 (maybe even 2.0), Microsoft has been saying that to write for Windows you need to know the Software Development Kit. (SDK) A few years ago, along with the release of Visual Basic MS introduced an SDK which was designed to produce a special type of Dynamic Load Library. (DLL) They called this SDK, the Control Development Kit or CDK. With this kit you are able to extend the application development toolkit provided with Visual Basic. What the kit produced when linked into your C programs was a DLL with a number of special features unique to Visual Basic and a few variations of C++. The developer puts a .VBX file extension on it but for all intents and purposes it was still a DLL.

Custom controls extend the VB Toolbox. Your custom control can use the standard properties, events, and methods built into Visual Basic, or it could introduce some totally new ones. You decide how your control is displayed and how its method behave. This article isn't going to teach you how to write a Custom Control, but rather how to use VBAPI_, one Pascal interface to the CDK. There are many books available describing how to write Custom Controls in C. This article will enable you produce a VB compliant control using the language we all love.

The Steps

I don't consider myself a Windows expert. I'm in the process of learning about Windows and that in itself is a huge undertaking. What I hope to do is explain what I went through getting my first VBX to work correctly and what files I needed to place in my uses to interface with the VBAPI.OBJ file which I extracted from the VBAPI.LIB which came with Visual Basic Pro 3.0.

Before I discuss the special properties that turn a DLL into a Custom Control and what must be done to make it a Visual Basic VBX, let me explain the project we'll be undertaking here. In the CDK, Microsoft has graciously given blanket permission to use their examples in any fashion we may see fit. Microsoft has not, however, turned loose the CDK itself, so... if you don't have the CDK or a copy of VB3.0 Professional then the following code will be of little use to you. If you're interested in turning your latest DLL project into a valuable 3rd party control then I suggest you look into getting Visual Basic. In this article I will be explaining how to translate the basic example in the CDK, PUSH.VBX to Pascal. This example subclasses a Windows push button and makes it your own.

The five steps necessary to write a VBX are:

1. Create your UP, Down, EGA and MONO tool buttons. These should be 28 X 28 to fit on the VB toolbar. I have included the bitmaps I used to create my version of PUSH in the files on this issue's disk.
2. Define your Property list. Of course to do this you have to have a complete design of what information your system needs.
3. Define your Event list. Same as number 2. In addition you'll have to be sure the properties you pass back and forth are in VB strings and not Pascal or null terminated string. VBAPI includes routines to convert null strings back and forth to VB strings.
4. Code your Control just like you would if it were a DLL. What won't work in a DLL, won't work in a VBX. Other than that, be flexible, remember that the user is (usually) providing all of your positioning data, the fonts, colors, text, etc. You merely take that information and tie it together in the context of the Control.
5. Change the extension to .VBX. You really don't have to do this step since VB and Windows doesn't care but the customer is used to including VBX's in their projects, and besides, DLL isn't in the default selection list when adding to a project.

Pascal Custom Control File

Thanks go to Microsoft for allowing unlimited use of the examples in the CDK. Without them I'd have had a much harder time developing my controls and producing this article.

The key to writing a control is in the interface between the program and the CDK library. This interface was made available on the BPASCAL forum on COMPUSERVE. Unfortunately I had no name to associate it with and it was released to the public domain. I freely give you that interface to experiment with. To use it you'll have to extract the VBAPI.OBJ file from the VBAPI.LIB provided in the SDK. Numerous library management tools exist in the various languages for that purpose.I'm not going to go into a long explanation about what is in the interface file. Most of it is self explanatory after you examine the CDK documentation. It merely translates the C data types and VB procedure calls into Borland Pascal. One warning however, if a routine is provided in the VBAPI then use that rather than a standard Windows API. This also allies to the messages passed between VB and your control. Many times they appear to be equal but in fact VB won't respond properly if the wrong one is used.

The program

So let's go through our sample VBX in Pascal, piece by piece. The listings show all of the code used, but remember it is also

on the disk, along with the supporting files too.
The first thing to do in include the VBAPI_ unit in you r uses list:

```
library PasPush;
{$R push.RES}
{$D Micro System Solutions - MS VB3.0 Push Demo}
uses
   wintypes,
   winprocs,
   vbapi_,
   strings;
```

This Unit provides translation from the VBAPI.LIB (OBJ) in the CDK and at the same time it translates from VB paradigm to the Borland T(object) paradigm
Next come some ID definitions (see Listing 1). Each of the bitmaps used in the control have to be created by and placed in the .RES file. The easiest way is, of course, to use the Resource Workshop. The IDBMP_Push bmp is the default used for the unselected picture of the bitmap and is the one used in the VB toolbar in the unselected mode. The IDBMP_PushDOWN is the selected bitmap and is only used in the toolbar when this tool is selected. The IDBMP_PushEGA & MONO are used for EGA and non-color screens and are produced in black and white only. Note that Autobeep in the TPush record is the only non standard data passed between the control and the VB Form.

```
{Toolbox bitmap resource IDs. }
const
IDBMP_Push          =  8000;
IDBMP_PushDOWN     = 8001;
IDBMP_PushMONO     = 8003;
IDBMP_PushEGA  = 8006;

// Standard Error Values}
ERR_None = 0;
ERR_InvPropVal    = 380;
{ Error$(380) = "Invalid property value"}
{   Procedure Declarations }
{   Global Variables and Constants }
{   Push control data and structs }
type
PPush = ^TPush;
TPush = record
    AutoBeep: Bool;
end;
```

Listing 1

The Properties List

The Properties list (See listing 2) is the technique used to communicate back and forth between the Visual Basic program and the Custom Control. To support a property, you declare it in the property list.

```
//---------------------------------------------------------------------------}
const
IPROP_Push_NAME   = $0000;
IPROP_Push_INDEX = $0001;
IPROP_Push_PARENT = $0002;
IPROP_Push_BACKCOLOR = $0003;
IPROP_Push_LEFT = $0004;
IPROP_Push_TOP = $0005;
IPROP_Push_WIDTH = $0006;
IPROP_Push_HEIGHT = $0007;
IPROP_Push_ENABLED = $0008;
IPROP_Push_VISIBLE = $0009;
IPROP_Push_MOUSEPOINTER = $000A;
IPROP_Push_CAPTION = $000B;
IPROP_Push_FONTNAME = $000C;
IPROP_Push_FONTSIZE = $000D;
```

```
IPROP_Push_FONTBOLD = $000E;
IPROP_Push_FONTITALIC = $000F;
IPROP_Push_FONTSTRIKE = $0010;
IPROP_Push_FONTUNDER = $0011;
IPROP_Push_DRAG = $0012;
IPROP_Push_DRAGICON = $0013;
IPROP_Push_TABINDEX = $0014;
IPROP_Push_TABSTOP = $0015;
IPROP_Push_TAG = $0016;
IPROP_Push_AutoBeep = $0017;

AutoBeepName : array[0..8] of Char = 'AutoBeep'#0;

Property_AutoBeep: tPROPINFO   = (
npszName :   tOffset(@AutoBeepName);
fl :  DT_Bool or PF_fGetData or
      PF_fSetData or
      PF_fSaveData;
offsetData       :   0;
infoData         :   0;
dataDefault          :   0;
npszEnumList :   0;
enumMax          :   0   );
const
PropListPush : array[0..24]of         ofsPPROPINFO = (
pPROPInfo_STD_CTLNAME,
PPROPINFO_STD_INDEX,
PPROPINFO_STD_PARENT,
PPROPINFO_STD_BACKCOLOR,
PPROPINFO_STD_LEFT,
PPROPINFO_STD_TOP,
PPROPINFO_STD_WIDTH,
PPROPINFO_STD_HEIGHT,
PPROPINFO_STD_ENABLED,
PPROPINFO_STD_VISIBLE,
PPROPINFO_STD_MOUSEPOINTER,
PPROPINFO_STD_CAPTION,
PPROPINFO_STD_FONTNAME,
PPROPINFO_STD_FONTSIZE,
PPROPINFO_STD_FONTBOLD,
PPROPINFO_STD_FONTITALIC,
PPROPINFO_STD_FONTSTRIKE,
PPROPINFO_STD_FONTUNDER,
PPROPINFO_STD_DRAGMODE,
PPROPINFO_STD_DRAGICON,
PPROPINFO_STD_TABINDEX,
PPROPINFO_STD_TABSTOP,
PPROPINFO_STD_TAG,
ofsPPropInfo(@Property_AutoBeep),
{ point to non-standard property}
0);
```

Listing 2.

Standard properties are those which are supported by the VB runtime.   Most of the properties in this list are standard Visual Basic properties. Non standard are those you wish to support and for which you have written supporting code.   The one non-standard property in this list is the beep property which will be used to provide a sound to the button.

The tPROPINFO structure defines the data and the processing used for the non-standard property.   In this case the property name, used in the VB property list is located at tOffset@AutoBeepName (see above).   The field is boolean and the VB runtime will get the data (PF_fGetData) directly from the record structure pointed to by the model structure. VB will also put the data (PF_fSetData) directly into the save structure. PF_fSaveData tells VB to get the value from the record structure and write it disk when the Form is saved.   Additional (and different) flag values can make VB act differently to different data.

The Event List

The Event list (See Listing 3) is the method by which the Custom Control interfaces with the VB program. The Event driven interface is what makes VB such a powerful development environment. The VB run-time will handle many standard events for you, while still giving you the option to override VB. In addition to the standard events you can add and fire your own events based on what your control is doing and what it needs from the calling VB program.

In the following list are three of the most rudimentary events required for a Custom Control. The *Click* event which is required for an object to gain the focus, and the *DragDrop/DragOver* events which allow the object to be placed and moved about the form. We'll see these and others in operation as we well.

The event name, *EventClick Name*, is made by VB into the subroutine / function name, for example:

Sub PasButton1_Click(ButtonCaption as String) VB uses the event parameter EventClickParm, include parameters in the subroutine call.

```
type
TParams = record
ClickString:        HLStr;
Index : Pointer;    { Reserve space for index parameter to array ctl}
end;

{ Event list   Define the consecutive indicies for the events
const
EVENT_PUSH_CLICK           = 0;
EVENT_PUSH_DRAGDROP        = 1;
EVENT_PUSH_DRAGOVER        = 2;
EVENT_PUSH_GOTFOCUS        = 3;
EVENT_PUSH_KEYDOWN         = 4;
EVENT_PUSH_KEYPRESS        = 5;
EVENT_PUSH_KEYUP                                          = 6;
EVENT_PUSH_LOSTFOCUS       = 7;

Event procedure parameter prototypes
Parms_SD : array[0..0] of word = (ET_I2);                 {Integer data type }

EventClickName : array[0..8] of Char =               'Click'#0;
EventClickParm: array[0..24] of char = 'ButtonCaption as String'#0;

Event_Click: tEVENTINFO   = (
npszName:           tOffset(@EventClickName);
cParms:             1;
cwParms:            2;
npParmTypes:  tOffset(@Parms_SD);
npszParmProf:  tOffSet(@EventClickParm);
fl:                                 0
);

EventListPush: array[0..8]of ofsPEVENTInfo = (
ofsPEventInfo(@Event_Click),
PEVENTINFO_STD_DRAGDROP,
PEVENTINFO_STD_DRAGOVER,
PEVENTINFO_STD_GOTFOCUS,
PEVENTINFO_STD_KEYDOWN,
PEVENTINFO_STD_KEYPRESS,
PEVENTINFO_STD_KEYUP,
PEVENTINFO_STD_LOSTFOCUS,
0);
```

```
EventListPush: array[0..8]of ofsPEVENTInfo = (
ofsPEventInfo(@Event_Click),
PEVENTINFO_STD_DRAGDROP,
PEVENTINFO_STD_DRAGOVER,
PEVENTINFO_STD_GOTFOCUS,
PEVENTINFO_STD_KEYDOWN,
PEVENTINFO_STD_KEYPRESS,
PEVENTINFO_STD_KEYUP,
PEVENTINFO_STD_LOSTFOCUS,
0
```

Listing 3

```
function PushCtlProc( Control: HCtl;Wnd: HWnd;   Msg,   WParam: Word;LParam: LongInt ) : LongInt; export;
var
 Params:          TParams;
 StrBuf:          array[0..19]of char;
 Caption:         Integer;
 error:           Err;
 tmpStr:          PChar;
   Push:          PPush;
begin
 Push := PPush(VBDerefControl( Control));
 case Msg of
    VBM_MNEMONIC,
    VBN_COMMAND: begin
       if Msg = VBM_MNEMONIC then                    { Act like a click}
LParam := MAKELONG( 0,        BN_CLICKED);
case HIWORD(LParam) of
   BN_CLICKED: begin
   Caption := GetWindowText( Wnd, StrBuf, 20);
   Params.ClickString := VBCreateHlstr( @StrBuf, Caption);
   if Push^.AutoBeep then
                              Messagebeep(0);
                             error := VBFireEvent( Control,EVENT_Push_Click,@Params);
                      VBDestroyHlstr( Params. ClickString);
   end;
   end;
   PushCtlProc := 0;
   exit;
   end;
VBM_SETPROPERTY:
case WParam of
IPROP_PUSH_CAPTION:
   { To avoid a Windows problem, make sure text is under 255 bytes:}
   if (lstrlen(PChar(LParam)) > 255) then
                             PChar(LParam)[255] := #0;
      end;
   end;
{// Default processing:}
PushCtlProc := VBDefControlProc(Control, Wnd, Msg, WParam, LParam);
end;
```

Listing 4

Control Procedure
This is the main control procedure for the Custom Control (See Listing 4). There can be multiple control procedure if your Custom Control implements multiple control objects. The model structure (described later) will reference this procedure.

The Model Structure
This defines the control model (using the event and property structures). It is defined after the control proc because it must reference it in the parameter list. This is probably the most important structure in the program. This is used to tell the Visual Basic runtime what you are and where your various internal structures are located. See Listing 5

Register Custom Control
The control is registered by the function *VBINITCC* (See Listing 6), which is called by VB when the custom control DLL is loaded for use. After this the exports are defined, and that's it!

Finishing Off
Listing 7 shows the Visual Basic make file required, and Listing 8 is the VB form.
That's all there is to creating a VBX in Borland Pascal. In the next installment, I'll take the basic code I developed here and turn it into a really useful control I call DiskTool. It gets information from the drives on your system, information VB can't directly get, and passes that information to your Visual Basic form. Two interesting features in the control are the ability to include an enumerated field in the property list for the drive type, and the ability to produce a development time only tool.

```
const
ModelDefCtlName:   array[0..8] of Char = 'PasPush'#0;                   { default control name prefix}
  ModelClassName: array[0..14] of Char = 'PasPushButton'#0;             { Visual Basic class name}
  ModelParentClassName: array[0..8] of Char = 'Button'#0;               { Parent window class if subclassed}
  modelPush: TMODEL = (
usVersion:  VB_VERSION;                         { VB version used by control}
fl:   MODEL_fFocusOk or MODEL_fMnemonic;   { Bitfield structure}
ctlproc: TFarProc(@PushCtlProc);          { The control proc.}
fsClassStyle:    cs_VRedraw or cs_HRedraw;    { window class style}
flWndStyle:      BS_PUSHBUTTON;           { default window style}
cbCtlExtra:      sizeof(TPush);           { # bytes alloc'd for HCTL structure}
idBmpPalette:    IDBMP_Push;                   { BITMAP id for tool palette}
DefCtlName: tOffset(@ModelDefCtlName);   { default control name prefix}
ClassName: tOffset(@ModelClassName);         { Visual Basic class name}
ParentClassName:  tOffset(@ModelParentClassName);                       { Parent window class if subclassed}
proplist:        ofs(PropListPush);       { Property list}
eventlist:       ofs(EventListPush);      { Event list}
nDefProp:        0;                            { index of default property}
nDefEvent:       0 );                     { index of default event}
```

Listing 5

```
function VBINITCC(usVersion: Word;    fRunTime: Boolean): Boolean; export;
begin
  VBINITCC := VBRegisterModel(           HInstance, modelPush);
end;

exports
  VBINITCC index 2,
  PushCtlProc index 3;

begin
end;
```

Listing 6

```
TPWPUSH.FRM
TPW_PUSH.VBX
ProjWinSize=152,402,248,215
ProjWinShow=2
```

Listing 7

Listing 8

```
VERSION 2.00
Begin Form Form1
Caption = "Visual Basic VBX in Borland Pascal"
   ClientHeight     =    1605
   ClientLeft       =    1710
   ClientTop        =    3645
   ClientWidth      =    6105
   Height           =    2010
   Left             =    1650
   LinkTopic        =    "Form1"
   ScaleHeight      =    1605
   ScaleWidth       =    6105
   Top              =    3300
   Width            =    6225
   Begin PasPushButton PasPush2
      AutoBeep          =    0    'False
      Caption           =    "E&xit"
      Height            =    375
      Left              =    2760
```

```
        TabIndex         =   2
        Top              =   960
        Width            =   1815
    End
    Begin PasPushButton PasPush1
        AutoBeep         =   0   'False
        Caption          =   "&Test me"
        Height           =   375
        Left             =   480
        TabIndex         =   1
        Top              =   960
        Width            =   2055
    End
    Begin TextBox Text1
        Height           =   495
        Left             =   480
        TabIndex         =   0
        Text             =   "Text1"
        Top              =   240
        Width            =   2175
    End
End
Option Explicit
Sub PasPush1_Click (ButtonCaption As String)
Text1 = PasPush1.Caption
End Sub

Sub PasPush2_Click (ButtonCaption As String)
Unload Me
End
```

Fred Hill is president of Micro System Solutions. His company produces DOS and Windows applications and tools in Borland Pascal 7.0 and Visual Basic 3.0.