# Creating VCL-Based Windows in DLLs

**by Charlie Calvert**

It is easy to place a VCL object in a DLL, as shown by the PICDLL project available from Compuserve. This DLL includes a Delphi form that can be called from a standard Delphi program.

The form enables the user to page through a selection of pictures. It isn't particularly useful, but it is pretty to look at and easy to construct, and broadly outlines the syntax used in placing Delphi objects in DLLs. The code for the unit is shown in Listing 3.

**Listing 3. The code from the PICS unit has one procedure, declared with the export directive.**

```
unit Pics;

{ Program copyright (c) 1995 by Charles Calvert }
{ Project Name: RUNDLL }

interface

uses
 SysUtils, WinTypes, WinProcs,
 Messages, Classes, Graphics,
 Controls, Forms, Dialogs,
 ExtCtrls, TabNotBk;

type
 TSpacePict1 = class(TForm)
   TabbedNotebook1: TTabbedNotebook;
   Image1: TImage;
   Image2: TImage;
   Image3: TImage;
 private
   { Private declarations }
 public
   { Public declarations }
 end;

var
 SpacePict1: TSpacePict1;

procedure ShowPictures(Handle: THandle); export;
```

```
implementation

{$R *.DFM}

procedure ShowPictures(Handle: THandle);
begin
 Application.Handle := Handle;
 SpacePict1 := TSpacePict1.Create(Application);
 try
   SpacePict1.ShowModal;
 finally
   SpacePict1.Free;
 end;
end;

end.
```

The ShowPictures procedure is not a method of TSpacePict. Instead, it is a simple routine declared in the interface of the unit with the export directive:

```
procedure ShowPictures; export;
```

Here is what the routine looks like:

```
procedure ShowPictures(Handle: THandle);
begin
 Application.Handle := Handle;
 SpacePict1 := TSpacePict1.Create(Application);
 try
   SpacePict1.ShowModal;
 finally
   SpacePict1.Free;
 end;
end;
```

The procedure allocates memory for a TSpacePict object, calls its ShowModal method, and finally destroys the object. When you want to use the TSpacePict object from another program, this is the only routine you need to call.

Note that ShowPictures has one parameter, which is the Application.Handle of the program that calls the procedure. It is not absultely necessary for you to pass it in, but you won't get a real modal

dialog if you don't pass in the Application.Handle of the calling program and assign it to the Application.Handle of the DLL. If you don't want to create a modal dialog then the parameter isn't absolutely necessary.

There is a try..finally block wrapped around the calls to Show and Free. This is the correct way to handle memory allocation, and you should get in the habit of always handling allocations this way, whether they are in a DLL or not.

The PICS unit is one of three units used by the FRACTDLL library. Listing 4 shows the source code for the main module of this DLL.

**Listing 4. The FRACTDLL exports three routines; each routine is found in a separate unit.**

```
library Fractdll;

uses
 Pics in 'PICS.PAS' {SpacePict1},

exports
 ShowPictures index 1;

{$R *.RES}

begin
end.
```

The exports clause for the FRACTDLL unit, as shown here, lists only one routine. (More routines are present in the included example program.) The ShowPictures routine is exported with an index of 1. If you take the code in the PICS unit and add in the exports statement shown here, you can see how simple it is to export a Delphi form from a DLL. Since DLLs can be called from Paradox, C++, dBASE, and many other languages, it is possible to take advantage of Delphi's ease of use and extraordinary technical capabilities to quickly create elaborate forms that can be used by a wide range of programs.