The Delphi CLINIC

Edited by Brian Long

Problems with your Delphi project?

Just email Brian Long, our Delphi Clinic Editor, on 76004.3437@compuserve.com or write/fax us at The Delphi Magazine

Service Not Good Enough

Q The Delphi online help and documentation on the subject of DDE says that a Delphi DDE server application's service name, or application name, will match that of the project without the extension. Can this be altered (ReportSmith does not conform to this rule)?

A Well, ReportSmith is not written in Delphi (perhaps it wouldn't be so big and slow if it were), but that's not the point. Indeed this rule can be altered. The DDE components make use of a component called `DDEMgr`, of the undocumented component type `TDDEMgr`, which has a property called `AppName`, used to define the DDE service. Fortunately, `DDEMgr` is declared in the interface section of the `DDEMan` unit, so something like:

```
DDEMgr.AppName :=
  'NewDDEServiceName';
```

in your server form's `OnCreate` handler should do the trick.

Combo Woes

Q The `TDBLookupCombo.OnClick` event doesn't fire if I click on the drop down list. However `OnClick` does work in this way for the other combo box components. What's wrong with it?

A The `TDBLookupCombo` is not a real combo box, but is made up from, among other things, an edit box and a list box. This is also why a `TDBLookupCombo` on a form with a `FormStyle` of `fsStayOnTop` doesn't show its drop down part at all: the list box used by the VCL stays behind the stay-on-top form.

Although the list box part of this fake combo does have an `OnClick` event handler, it is only used to determine when to hide the list box.

We can improve the situation by extending the functionality of this listbox's `OnClick` event and causing it to also call the `TDBComboBox`'s `OnClick` in a new component. Listing 1 shows how. The constructor for `TNewDBLookup` searches through its own component list until it finds a `TPopupGrid` (a class based on a `TDBLookupList`), then replaces its `OnClick` handler with a new method, saving the old one first. The new handler calls the old one, then calls the previously uncalled `OnClick` handler.

Microsoft Products And Floating Point

Q I have a DLL written in a Microsoft C compiler which exports a function returning a `Double`. When I try and call it, I don't get the values I expect. Is there a compatibility problem between Borland and Microsoft products?

A That's exactly what the problem is, and it usually shows up with functions that return floating point values. For example, a Visual Basic application will have trouble getting a function in a Delphi DLL to return a floating point value (hereafter referred to as a float for brevity), as will an Excel or Microsoft C application.

Regardless of what compiler options are used, Borland products cause functions to return floats on the NDP (Numeric Data Processor, or 80x87 co-processor) stack.

➤ *Listing 1*

```
unit Newcomb;
interface
uses
  SysUtils, WinTypes, WinProcs, Messages, Classes,
  Graphics, Controls, Forms, Dialogs, StdCtrls, DBLookup;
type
  TNewDBLookup = class(TDBLookupCombo)
  private
    FOldGridClick: TNotifyEvent;
  protected
    procedure NewGridClick(Sender: TObject);
  public
    constructor Create(AOwner: TComponent); override;
  end;
procedure Register;
implementation

constructor TNewDBLookup.Create(AOwner: TComponent);
var  Loop: Word;
begin
  inherited Create(AOwner);
  for Loop := 0 to Pred(ComponentCount) do
    if Components[Loop] is TPopupGrid then
      with Components[Loop] as TPopupGrid do begin
        FOldGridClick := OnClick;
        OnClick := NewGridClick;
        Break;
      end;
end;

procedure TNewDBLookup.NewGridClick(Sender: TObject);
var  FOnClick: TNotifyEvent;
begin
  if Assigned(FOldGridClick) then
    FOldGridClick(Self);
  FOnClick := OnClick;
  if Assigned(FOnClick) then
    FOnClick(Self);
end;

procedure Register;
begin
  RegisterComponents('Samples', [TNewDBLookup]);
end;
end.
```

Microsoft products don't always work like this and so, generally, the advice is to make cross-vendor applications work with functions that return pointers to floats instead, since pointers are returned the same by both Borland and Microsoft compilers. Sometimes it is not possible to do this as the EXE or DLL may be a commercially purchased item whose source is not available, and so we need a better solution, which of course will rely on knowing what Microsoft compilers do when they don't use the NDP stack. We'll cater for both possibilities, where you may be writing the DLL in Delphi for an MSC app, or an app in Delphi for an MSC DLL.

When compiled with Pascal calling conventions, Microsoft C code that calls a function returning a float declares a hidden local variable, and passes the offset as an extra parameter to the subroutine. The subroutine stores the intended return value in this argument, whose address it builds up from the stack segment and the given offset. What the routine actually returns is the address of this variable. At least that is according to MS documentation: I have found that it only returns the offset, but it makes little difference in practice.

When compiled with C calling conventions, MSC code returns `Extended` values on the NDP stack, as does Delphi, but `Singles` and `Doubles` are returned via a global variable. The function stores the return value in a variable (which happens to be called `__fac`, the floating point accumulator) and returns its address.

If we consider an MSC written application that can call a DLL which defines routines that are supposed to look like Listing 2, we would actually need to implement them as shown in Listing 3.

If we now take the other angle, if we have an MSC written DLL, that we are told contains effectively what Listing 2 shows, to call it we might normally expect something like Listing 4, but instead would have to use what's in Listing 5.

So you can check these solutions out, I have included various files on this month's disk. Firstly there are the source and binaries for an MSC generated EXE and DLL called MSAPP.EXE and DBLDLL.DLL. Also there are projects for the two Delphi plug-in replacements as MSAPP.DPR and DBLDLL.DPR. They have conditional compilation directives to cater for Borland or MS-style code generation, but are set up for MS by default.

➤ *Listing 2*

```
library Dbldll;
uses  WinProcs;
function TestFloatPascal(
  D: Double): Double; export;
begin
  Result := D;
end;
function TestFloatCDecl(
  D: Double): Double; cdecl;
  export;
begin
  Result := D;
end;
exports
  TestFloatPascal index 2,
  TestFloatCDecl  index 3;
begin
end.
```

➤ *Listing 3*

```
library Dbldll;
uses  WinProcs;
type  PDouble = ^Double;
var   __fac: Double; { Global variable for floating point operations }
function TestFloatPascal(D: Double; Offset: Word): PDouble; export;
begin
  Result := Ptr(SSeg, Offset); { Return address of result on stack }
  Result^ := D; { Store result in stack at given offset }
end;
{ If the parameter is an Extended (long double), this change won't be
  necessary - MSC returns it in the same way as Delphi - on the NDP stack }
function TestFloatCDecl(D: Double): PDouble; cdecl; export;
begin
  Result := @__fac; { Return address of result in DLL data segment }
  Result^ := D; { Store result in DLL variable }
end;
exports
  TestFloatPascal index 2,
  TestFloatCDecl  index 3;
begin
end.
```

➤ *Listing 4*

```
program Msapp;
uses SysUtils, Dialogs;
procedure WinMain;
begin
  ShowMessage(FloatToStr(
    TestFloatPascal(2.5)));
  ShowMessage(FloatToStr(
    TestFloatCDecl(2.5)));
end;
begin
  WinMain;
end.
```

➤ *Listing 5*

```
program Msapp;
uses SysUtils, Dialogs;
type PDouble = ^Double;
function TestFloatPascal(D: Double; Offset: Word): PDouble;
  far; external 'DBLDLL' index 2;
function TestFloatCDecl(D: Double): PDouble; cdecl;
  far; external 'DBLDLL' index 3;
procedure WinMain;
var  Temp: Double;
begin
  ShowMessage(FloatToStr(TestFloatPascal(2.5, Ofs(Temp))^));
  ShowMessage(FloatToStr(TestFloatCDecl(2.5)^));
end;
begin
  WinMain;
end.
```

Pointerless DLL Access

Q I am interfacing to a C DLL – I have to call a routine and define a routine to be called

compatible with some C prototypes, but am being forced to use the pointer symbols `^` and `@` and pointer types (see Listing 6). I thought Delphi allowed us to avoid pointers in most cases?

A Take advantage of the pass-by-reference modifiers `var` and `const`. If you will be modifying the value, use `var`, if not use `const`. Passing by reference is implemented by passing the *address* of variables – in other words, pointers (but we don't have to use pointer notation). See Listing 7.

Setting Properties En Masse

Q If I make a `TList` or array of components including, for example, a button and a menu item, I am unable to easily set certain properties such as `Enabled` or `Caption` without checking the types in a conditional statement. It seems that some similarly named properties have not been added in a common ancestor and so live at different offsets in memory, and GPFs prevail without full checks. Is there a way to simplify this?

A We can take advantage of the run-time type information (RTTI) stored in the objects and some RTTI-accessing functions in the undocumented `TypInfo` unit.

There are two applications on this issue's disk to demonstrate this. The first, MANY.DPR, defines a routine (written by Roy Nelson at Borland) to set an explicit property, `Enabled`, to a given value. The routine `ActivateControls` is shown in Listing 8, followed by an example of how to call it.

A more generic routine (based on Roy's code) to set a given named ordinal, floating point or string property to any value, is used in MANY2.DPR and given in Listing 9, with an example call.

Tracing SQL

Q Is there a way to see the SQL commands that get sent to my database server when using a `TTable` component?

A Indeed there is. A peculiarly undocumented setting in WIN.INI will cause Windows debug strings to be generated each time the BDE performs an operation on an SQL database. In fact the setting is interpreted by each of the SQL Link drivers and the ODBC socket (which used to be known as the Idapter).

Add an `SQLTrace` entry to the `IDAPI` section in WIN.INI. The value should be negative (`-1`) to see all operations with the SQL target (SQL preparations, executions, errors, statements, connections, BLOb I/O and miscellaneous), or positive (1) to see just the SQL expressions (SQL prepare operations). A zero turns the trace off.

```
[IDAPI]
DLLPATH=C:\IDAPI
CONFIGFILE01=C:\IDAPI\IDAPI.CFG
SQLTRACE=1
```

Sample output from the notification handler in my *Callbacks in Windows and The BDE: Part 3* article from this issue (which traps, among other things,

➤ *Listing 6*

```
{ C routine declarations:
  void _pascal FunctionToCall(unsigned int *Param);
  void _pascal FunctionToBeCalled(long double *Param); }
type PExtended = ^Extended;
var  W: Word;
     E: Extended;
procedure FunctionToCall(Param: PWord); external 'CDLL';
procedure Button1Click(Sender: TObject);
begin
  W := StrToInt(Edit1.Text);
  FunctionToCall(@W);
end;
procedure FunctionToBeCalled(Param: PExtended); export;
begin
  E := Param^;
end;
```

➤ *Listing 7*

```
{ C routine declarations: see Listing 6 }
var W: Word;
    E: Extended;
procedure FunctionToCall(var Param: Word); external 'CDLL';
procedure Button1Click(Sender: TObject);
begin
  W := StrToInt(Edit1.Text);
  FunctionToCall(W);
end;
procedure FunctionToBeCalled(const Param: Extended); export;
begin
  E := Param;
end;
```

➤ *Listing 8*

```
procedure ActivateControls(SetTo: Boolean;
  const ControlsToChange: array of const);
var I: integer;
    PropInfo: PPropInfo;
begin
  for I := Low(ControlsToChange) to High(ControlsToChange) do
    with TVarRec(ControlsToChange[I]) do
      { Sanity check to see if it is an object }
      if VType = vtObject then begin
        PropInfo := GetPropInfo(VObject.ClassInfo, 'Enabled');
        if Assigned(PropInfo) then
          SetOrdProp(VObject, PropInfo, LongInt(SetTo));
      end;
end;
{...}
ActivateControls(False, [Button1, Edit1, About1]);
```

Windows debug messages) is shown in Figure 1, where `SQLTrace` has been set to `1`.

Symbol File Error

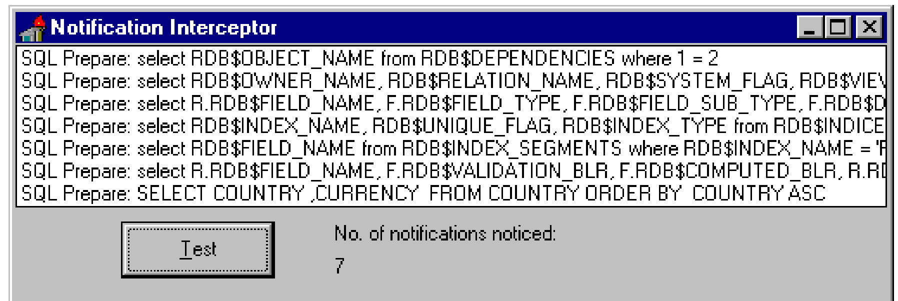
Q When I open some projects I get an *"Error reading symbol file"* message. How can I get rid of it, and why does it come up?

A There is an option in the `Autosave options:` group of the `Options|Environment` dialog on the `Preferences` page in the Delphi IDE, allowing you to save the desktop. This is designed to allow the layout of windows and content of history lists etc to be saved for the next time you open a project. When the option is checked, the `Desktop contents:` radio buttons on the left come into play. Most enlightened people will opt for `Desktop only` (save a .DSK file), but the default is `Desktop and Symbols` (save a .DSK and a .DSM file).

The .DSK file (about 1kb) is a Windows .INI file with sections storing information about the environment. The .DSM file (usually at least 500k) is a dump of the symbol table Delphi generates in memory when compiling your project. If the symbols are saved, when you reload your project you can use the Object Browser etc without being forced to re-compile your project first. In short, a few seconds benefit (well, not long anyway) costs you half a megabyte per project. If a .DSM file is being saved, a section like this is added to the .DSK file.

```
[Symbols]
SymbolFile=C:\DELPHI\PROJECT1.DSM
ExecName=C:\DELPHI\PROJECT1.EXE
```

If you move a project to a new directory, or take projects from other people, the chances are that the .DSM file will not be kept, but often the small .DSK file is. When Delphi opens the project and sees the `Symbols` section in the .DSK file, it tries to open the .DSM file. Any problem, like the file not being there, gives the error. The easiest solution is to delete the .DSK file.



➤ *Figure 1*

Credit Where Credit's Due

Q I've heard of the 4 Delphi About dialog Easter Eggs. Does the product have any more hidden goodies?

A Indeed it does. As you say the Delphi About box (Alt-H, A) has four undocumented key sequences. Hold down the Alt key and type `VERSION`, `DEVELOPERS`, `TEAM` or `AND` (the latter only works with 256 or more colour screen drivers).

The About box in ReportSmith (including both the run-time and Data Dictionary applications) can yield some credits by Ctrl-clicking the icon with your right mouse button 16 times (do it slowly or your clicks will be interpreted as double-clicks – if the icon flashes on clicks 7 to 16, then you are doing it correctly.

The BDE Config's About box changes with Alt-S, WinSight's About box reacts to Alt-I, WinSpector doesn't have an About box, but if you restore it from its natural iconic state, Alt-I also does something. Database Desktop's About box gives an uninteresting IDAPI version number with Alt-I (or even just I). Shift-Z gives the normal credits. Is that enough to be getting along with?

Incidentally, a little bird told me a story about the keystrokes for the winking snap of the Danish Delphi chief architect Anders Hejlsberg pictured in a swimming pool. Ducks sit in water; Danes refer to Donald Duck as Anders And; to see Anders, you type AND.

Acknowledgements

Thanks to Roy Nelson of Borland for the cunning property changing routine, and to Steve Axtell from Borland for the lookup list and informative SQLTrace tips.

➤ *Listing 9*

```
procedure ChangeControls(const Prop: String;
  const SetTo, ControlsToChange: array of const);
var
  I: integer;
  PropInfo: PPropInfo;
  Obj: TObject;
begin
  for I := Low(ControlsToChange) to High(ControlsToChange) do
    if TVarRec(ControlsToChange[I]).VType = vtObject then begin
      Obj := TVarRec(ControlsToChange[I]).VObject;
      PropInfo := GetPropInfo(Obj.ClassInfo, Prop);
      if Assigned(PropInfo) then
        with TVarRec(SetTo[Low(SetTo)]) do
          case VType of
            vtInteger, vtBoolean, vtChar:
              SetOrdProp(Obj, PropInfo, VInteger);
            vtExtended:
              SetFloatProp(Obj, PropInfo, VExtended^);
            vtString:
              SetStrProp(Obj, PropInfo, VString^);
          end;
    end;
end;
{...}
ChangeControls('Enabled', [False], [Button1, Edit1, About1]);
```