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

And Do Dilly Dally On The Way

Q How do I instigate some kind of delay in my application?

A If you are using Delphi 2, there is a Win32 API called `Sleep` that takes one parameter: a number of milliseconds. The trouble with `Sleep` is that if it is called from the main program thread, the program effectively hangs for the duration of the delay. If you try and move the form, it won't (until the delay is over). Likewise, if you obscure the program, and then uncover it again, it won't repaint itself until the delay is up.

The other two approaches are applicable to Delphi 1 and 2, and initially suffer from the same problem as `Sleep`, but at least it can be overcome with these.

Listings 1 and 2 show the two approaches. The comments in the two listings show how to let your program still function during the delay. Note that one of these `Delay` routines is called in Listing 3.

Changing Locations

Q When my programs start up, their forms are always in the same location on the desktop, and the same size, as they are at design time. When I run NotePad, its location varies, as does its size. Is the this something we can easily mimic?

A Indeed. The form's `Position` property defaults to `poDesigned`, which tells it to use the design time location and size. A value of `poScreenCenter` ensures it is positioned in the centre of the screen, whereas `poDefaultPosOnly` means that its position will vary but its size will be the same, `poDefaultSizeOnly` means the size will be vary but the position will be fixed and lastly `poDefault` matches the behaviour seen with NotePad where the size and position are variable.

Incidentally, setting `BorderStyle` to `bsDialog` in Delphi 2 will stop this varying placement.

Tacky Termination

Q If I decide to terminate my program (using `Application.Terminate`) in my main form's `OnCreate` or `OnShow` event, the program does close, but I get a flicker of the main form on the screen before it goes. How do I stop this?

A In Delphi 1, terminate with `Halt` instead. This isn't as good as `Application.Terminate` but exit routines will get called. In Delphi 2, terminate with whatever you like, but set the new Delphi 2 `Application.ShowMainForm` property to `False` first. It seems this property was added so that OLE automation server applications could start without showing their main forms if desired when started by an OLE automation controller.

Changing The Default Exception Handler

Q When I get an exception in my program, firstly the debugger tells me about it (I know I can turn this off with the `Break on Exception` option) and then my program reports it. I don't like the way it reports it, so how do I change it?

A When the debugger tells you about an exception that's happened in your program it is trying to be helpful, allowing you to debug the problem. You can certainly turn this off by un-checking

➤ *Listing 1*

```
procedure Delay(MSec: Longint);
const MSecPerDay = 24 * 60 * 60 * 1000;
var   OldTime: TDateTime;
begin
  OldTime := Time;
  repeat
    { To resolve the hung program, use the following commented out lines
    Application.ProcessMessages;
    if Application.Terminated then
      Break }
  until Time >= OldTime + MSec / MSecPerDay;
end;
```

➤ *Listing 2*

```
procedure Delay(MSec: Longint);
var OldTime: Longint;
begin
  OldTime := GetTickCount;
  repeat
    { To resolve the hung program, use the following commented out lines
    Application.ProcessMessages;
    if Application.Terminated then
      Break }
  until GetTickCount >= OldTime + MSec;
end;
```

the option you mention on the Preferences page of Options | Environment (Delphi 1) or Tools | Options (Delphi 2).

To change the default exception handler, you must write an event handler for the Application object's OnException event. The trouble with this is that the Application object does not appear in the Object Inspector: it is only available at run time. This means that the event handler must be set up by hand. To do this requires three steps, as performed by the Object Inspector when it makes event handlers.

First, declare an appropriate method in a form class. In fact it can be in any class, but form classes are conveniently being manufactured by the environment anyway. To make the method "appropriate" it needs the right interface, which is described in the help for the event type. Take as an example the OnDestroy event of a form. Find and select it on the Events page of the Object Inspector and then press F1. The help says it's a property of type TNotifyEvent. Click on the help link for the type definition, which describes appropriate event handlers as procedures taking a Sender parameter of type TObject, where the procedures are defined inside an object type: ie they are methods.

If you look up OnException, it is of type TExceptionEvent and appropriate event handlers take two parameters (Sender and the exception to handle). Incidentally, when you type in an appropriate method declaration, make sure it's in the private or public section of the form class, not the section that Delphi maintains: you'll get into trouble if you use that bit.

Secondly, set up the implementation. This means copying the declaration into the form unit's implementation section and preceding the method name with the class name and a dot. Now follow it with begin and end and a semicolon. If this all sounds confusing, make an OnCreate handler for your form by double-clicking on the form's client area and examine the declaration and implementation you can see placed in the editor.

Thirdly, associate the method with the event. If you made a form OnCreate handler as suggested above, look at the Events page of the Object Inspector for the form. The name of the event handler method is listed next to the event. It's this that turns the method into an event handler. Without this, it is just a method. Now, given that the online help page we looked at earlier said that events are really properties, and we often change property values at run-time with assignments, it follows that we can also associate our custom method with Application.OnException with an assignment. Where we place the assignment depends on how long we want the event handler active, but this will often happen in the main form's OnCreate handler to make it active as long as possible.

These three steps reproduce what the Object Inspector does when it make an event handler. All that we've left to do now is put code inside the method we have made. Listing 3 shows a sample OnException handler: note it uses MessageDlg to tell the user about the error (the original one used ShowMessage). Also it indicates a General Protection Fault or Access Violation by temporarily writing Ouch! on the main form's caption bar. The EXCEPTS.DPR program on the disk shows the handler being used to trap three otherwise unhandled exceptions. Listing 4 shows the three event handlers that generate the errors.

Can't Re-Compile The VCL

Q Delphi 1 allows me to add the VCL source directory onto my unit search path in the Directories/Conditionals page of the Project Options dialog. Having done this, a simple recompilation is necessary to then allow me to go

➤ *Listing 3*

```
type
  TForm1 = class(TForm)
    ...
    procedure FormCreate(Sender: TObject);
  private
    procedure DoException(Sender: TObject; E: Exception);
    ...
  end;
...
procedure TForm1.DoException(Sender: TObject; E: Exception);
var S: String;
begin
{$ifdef Win32}
  if E is EAccessViolation then begin
{$else}
  if E is EGPFault then begin
{$endif}
    S := Caption;
    Caption := 'Ouch!';
    Delay(750);
    Caption := S
  end else
    MessageDlg(E.ClassName + ': ' + E.Message, mtError, [mbCancel], 0)
end;
procedure TForm1.FormCreate(Sender: TObject);
begin
  Application.OnException := DoException
end;
```

➤ *Listing 4*

```
procedure TForm1.Button1Click(Sender: TObject);
var P: PByte;
begin
  P^ := 0 {Generate a GPF or AV }
end;
procedure TForm1.Button2Click(Sender: TObject);
begin
  StrToInt('A') { A is not a valid integer}
end;
procedure TForm1.Button3Click(Sender: TObject);
begin
  { Invalid typecast }
  (Sender as TListBox).ItemIndex := 0;
end;
```

stepping into the VCL source. I tried this with Delphi 2 and didn't get very far. Even with the correct directory in the search path box, re-compiling or rebuilding does not go though the VCL source files. What's changed?

A Unit caching has been added, and it acts in a special way for the VCL source files. Basically, after you first compile a Delphi project in the IDE, the VCL units are loaded into memory. Each successive compilation makes use of these cached units. Unfortunately, despite adding the source path to the unit search path, the compiler refuses to replace the VCL units in the cache. After setting the search path accordingly, you must close Delphi 2, saving the changes to the project, and then restart it and reload the project. The rebuild or recompilation will now work.

OLE Method Lost

Q I am writing an OLE automation controller where I repeatedly refer to the OLE server's properties from within a `TTimer`'s `OnTimer` event handler. When the application is running under Windows 95 and I right click on the server's icon in the task bar, I get an exception each time my timer ticks, saying that the method or property I referred to does not exist. Why is this, when I know full well that it does?

A Under those circumstances, OLE2 does yield an error, but the one reported has an incorrect message. Whenever you call something from an OLE automation server, the server is interrogated for the dispatch id number of the properties or methods you refer to. The variant you use to represent the connection to the server holds an `IDispatch` object and that object's `GetIDsOfNames` method is called. When you refer to something that is not implemented in the server, that routine returns a `DISP_E_UNKNOWNNAME` error (number `$80020006`). When the task bar right click menu is up, that method returns an error with the snappy little name of

```
RPC_E_CANTCALLOUT_ININPUTSYNCCALL
```

(number `$8001010D`). This indicates that whilst the menu is up, Windows won't allow certain operations and so OLE calls are forbidden.

OLE does have support for generating error messages when an error value arises (you can pass the value to `OleCheck`: if it is between `$7FFFFFFF` and `$FFFFFFFF` an exception gets generated with the error message).

The OLE error that is given when you call an invalid automation property or method is *Unknown name*. Because this isn't very descriptive, the code in the `OLEAuto` unit generates a custom exception, with the message *Method xxxx is not supported by OLE object*. However this message is used regardless of the error number returned by `GetIDsOfNames`. The return value should be checked, and the custom message should only be used when an error of `DISP_E_UNKNOWNNAME` is detected.

To fix this problem, you can modify the `OLEAuto` unit, re-compile it and copy the unit into Delphi's LIB directory. The relevant routine to modify in the unit is the `GetIDsOfNames` procedure (search for `procedure GetIDsOfNames`), where you can see `Dispatch.GetIDsOfNames` being called at the end. Notice that the only check is against zero: if the return value is not zero, it generates the custom `EOleError` exception with a message as designated by the `SNoMethod` constant. Listing 5 is the code as it stands, Listing 6 is the modified version that works properly. Re-running the program with this change in place causes the error to be the more correct *An outgoing call cannot be made since the application is dispatching an input-synchronous call.*

Note that to get this new VCL unit compiled into your program without problem, refer to the previous Clinic item.

Credits Addendum

In February (Issue 6) I listed all the known Delphi 1 hidden credit/gang screens. Delphi 2 removes the colourful `Alt+AND` picture of Anders Hejlsberg, but another one has come to light. Invoke the BDE API help file and make sure you are looking at the `Index`, then choose `overriding defaults` and push the little button at the bottom left of the text. Now choose `credits`. This gives a list of those responsible for the BDE.

➤ *Listing 5*

```
procedure GetIDsOfNames(Dispatch: IDispatch; Names: PChar;
  NameCount: Integer; DispIDs: PDispIDList);
var
  ...
begin
  ...
  if Dispatch.GetIDsOfNames(GUID_NULL, @NameRefs, NameCount,
    LOCALE_SYSTEM_DEFAULT, DispIDs) <> 0 then
    raise EOleError.CreateResFmt(SNoMethod, [Names]);
end;
```

➤ *Listing 6*

```
procedure GetIDsOfNames(Dispatch: IDispatch; Names: PChar;
  NameCount: Integer; DispIDs: PDispIDList);
var
  ...
  Res: HResult;
begin
  Res := Dispatch.GetIDsOfNames(GUID_NULL, @NameRefs, NameCount,
    LOCALE_SYSTEM_DEFAULT, DispIDs);
  if Res = DISP_E_UNKNOWNNAME then
    raise EOleError.CreateResFmt(SNoMethod, [Names])
  else
    OleCheck(Res);
end;
```