Is Delphi Running?

by Brian Long

Here is a question that was sent to the *Delphi Clinic* not so long ago:

I have been searching high and low for examples of how to detect when my program is running from within the IDE and when it is not. So far, all I have found are references to things that I do not know much about. I know this is what a lot of developers do with shareware/trialware programs, allowing all functionality as long as the program is being launched from the IDE. I need similar functionality, but because I need to simulate my user's environment in code when running on my machine, but I don't want the application to do any simulation when it is out in the field on users' machines. Can you help?

This technique does indeed get used quite a lot. It is very convenient for Delphi component writers to offer their full software to people, with the restriction that it only works if Delphi is running as well. This saves on the time necessary to develop crippled versions of software, and allows potential purchasers to check out the full functionality on offer. Dave Jewell focused on this angle in Issue 43 with *Protecting Your Intellectual Property* in his *Beating the System* column.

➤ *Listing 1*

```
interface
const
  RunningInIDE: Boolean = False;
procedure Register;
implementation
procedure Register;
begin
  RunningInIDE := True
end;
```

However, as you can see from the question, this type of requirement is not only about providing trial versions of software. It can also involve having your application act differently when running from the IDE for other reasons. In the case in hand, the questioner is developing an application for a target environment of his client. Since he does not have all the software or hardware facilities that his client does, he wishes to simulate some of it, so that, as he develops the application, he can still test it out on his own machine.

The question itself has a number of variations. Let's take them one at a time.

Different Execution

How can I have some code that executes differently when installed in the Delphi IDE compared to when used in an application?

This may or may not apply to the questioner. The question relates to software that can be installed *into* Delphi, as opposed to being launched *from* Delphi. So, it relates to code that resides in design-time packages in Delphi 3 or later.

Incidentally, in truth all the packages that you create will be runtime packages, because they contain compiled code. The prime distinction between design-time and runtime packages is that runtime packages will be used by applications that you create, or indeed will be used by design-time packages. Design-time packages are those that will be used by an application created and compiled by Inprise, which is Delphi in our case.

So, to install some code into Delphi, you compile your unit into a design-time package (or Delphi's component library in the case of versions 1 and 2). Code to be installed into Delphi would include the implementation of some components, or property and component editors, or experts, or IDE add-in tools that can manipulate and customise the Delphi environment (see my article in Issue 27 on *Delphi 3 Add-In Packages*).

When Delphi loads the package, or component library DLL, it calls the `Register` routine from each unit, if one exists. So you can implement the `Register` routine to set some flag, as in Listing 1, which will then indicate if your code is running in the Delphi IDE, or in an independent application. Various other parts of your code can check the flag like this to decide where they are:

```
if RunningInIDE then
  { IDE version of code}
else
  { Other code}
```

Of course, if the code is part of a component class, this flag is pretty redundant since you can check whether `csDesigning` is a member of the `ComponentState` set property to find out the same information:

```
if csDesigning in
  ComponentState then
  { IDE version of code}
else
  { Other code}
```

Delphi Installed?

How can I tell if Delphi is installed on the computer my program is running on?

Delphi 1 uses an INI file in the Windows directory called DELPHI.INI. Checking for the existence of the file would do the job, although this in itself is not foolproof: Delphi 1 could have been deleted. Listing 2 takes the approach of reading something from the INI file that should be there. If it gets a sensible response, the INI file exists. The value read is

➤ *Listing 2*

```
function Delphi1Exists: Boolean;
begin
  with TIniFile.Create('DELPHI.INI') do
    try
      //If entry is not in INI file, this will return
      // a file name of '' which will not exist
      Result := FileExists(ReadString('Library', 'ComponentLibrary', ''))
    finally
      Free
    end
end;
```

```
function Delphi32Exists: Boolean;
var
  Reg: TRegistry;
  Keys, Values: TStrings;
  KeyLoop, ValueLoop: Integer;
const
  DelphiPath = 'Software\Borland\Delphi\';
begin
  Result := False;
  Reg := TRegistry.Create;
  Keys := TStringList.Create;
  Values := TStringList.Create;
  try
    Reg.RootKey := HKEY_LOCAL_MACHINE;
    if Reg.OpenKey(DelphiPath, False) and Reg.HasSubKeys
      then begin
      //There may be more than one Delphi section
      Reg.GetKeyNames(Keys);
      Reg.CloseKey;
      for KeyLoop := 0 to Keys.Count - 1 do
        if not Result and Reg.OpenKey(DelphiPath +
          Keys[KeyLoop], False) then
          try
            Reg.GetValueNames(Values);
            for ValueLoop := 0 to Values.Count - 1 do begin
              Result := (Pos('Delphi', Values[ValueLoop]) >
                0) and
                FileExists(Reg.ReadString(
                  Values[ValueLoop]));
              if Result then Break
            end;
          finally
            Reg.CloseKey
          end
    end
  finally
    Reg.Free;
    Keys.Free;
    Values.Free
  end
end;
```

➤ *Listing 3*

the path of the Delphi 1 component library. A test is also made for the presence of the component library to ensure Delphi 1 is still installed.

All 32-bit versions of Delphi store their details in the registry rather than INI files. Typically, the per-user details are stored in

```
HKEY_CURRENT_USER\Software\
  Borland\Delphi\X.0
```

where `X` is the major Delphi version number. However, if another user logs into Windows, this registry path may well not exist (as the user may not yet have run Delphi). So a more reliable approach is to check the per-machine details in

```
HKEY_LOCAL_MACHINE\Software\
  Borland\Delphi\X.0.
```

As well as existing regardless of the Windows user (remember one copy of Windows can have different users logging into it at different times), this has an additional advantage. There is a value under this key that gives the full path to the Delphi executable file. Unfortunately, the name of this value is inconsistent: `Delphi 2.0`, `Delphi 3`, `Delphi 4`. To check that Delphi really is installed, you can verify that the referenced file does indeed exist. Cue Listing 3 (which, incidentally, has only one outermost `try..finally..end` statement where it should really have three: this was simply for brevity). You will notice that it has to loop through the registry entries for any Delphi key that it finds, looking for one that starts with `Delphi` to overcome the inconsistency mentioned above.

```
type
  { Used by TDebugRec }
  TExceptionKind =
    (evNull, evRaise, evExcept, evFinally, evUnexpected, evTerminate);
  PDebugRec = ^TDebugRec;
  TDebugRec = record
    dhMagic1,
    dhZero,
    dhMagic2,
    dhHookProc,
    dhDebugHooked: Longint;
    dhKind: Word; { Use TExceptionKind enumerated type above }
    dhAddr,
    dhCookie,
    dhNameLen,
    dhName,
    dhMsgLen,
    dhMsg,
    dhWantException,
    dhDoneExcept: Longint;
  end;
const
  DebuggerHook  = $24; { Offset in DS of pointer to debugger data }
{ Checks if debugger is active (it swallows notifications) }
function DelphiDebuggerRunning: Boolean;
begin
  Result := (PrefixSeg <> 0) and
    (LoWord(PDebugRec(Ptr(DSeg, DebuggerHook)^)^.dhDebugHooked) <> 0);
end;
```

➤ *Listing 4*

IDE Running

How can I tell if Delphi is running or not?

To find out if the Delphi IDE is running, we need to identify whether any of the IDE windows are open on the desktop. You should refer to the aforementioned article by Dave Jewell for a variety of ways to accomplish this by searching for window captions.

Launch Party

How can I tell if Delphi launched my application?

There are a couple of approaches commonly used to answer this question. We will look at both of them.

One of the original efforts in this direction revolved around understanding the operation of Delphi 1's debugger. *The Revolutionary Guide To Delphi 2* (various authors, published by Wrox Press) discusses this subject in Chapter 15.

When the integrated debugger launches an application, it inserts some bytes in some internal data structures of the program to allow the debugger to be notified of certain things happening. An application can examine these data structures, and identify if the debugger was active when the program was launched. This *almost* equates to a test of whether the IDE launched your application. I say 'almost' because if integrated debugging is disabled the program will assume that the IDE is not running if using this test alone.

Testing for the debugger in Delphi 1 is woefully more involved than in 32-bit Delphi, but let's

struggle through it. The test relies upon a record structure whose only documentation is its use in the runtime library. The record is used to interface to a debugger when exception-related events occur, and various operations are required of it. It is referenced in the SysUtils unit, in addition to the EXCP.ASM exception handling assembler runtime library source module and the SE.ASM assembler include file.

A Pascal version of the assembler structure, along with a suitable function, is shown in Listing 4. So the address of a TDebugRec record is $24 bytes into the data segment, whose dhDebugHooked field (or at least the low byte of it) gives the game away.

If that all looks too excessive, then we can remove all the type definitions and cut to the chase. Listing 5 shows a much shorter version of the same thing. Also, Listing 6 shows the 32-bit Delphi equivalent: the System unit now defines a global variable, DebugHook, that surfaces this information.

On top of these approaches, you could also use Dave Jewell's suggestion for locating the parent process of your application. This will be a foolproof way of seeing whether Delphi launched it, but does require you to write markedly different logic for Windows NT and Windows 95/98, which could be troublesome.

So, where are we now? We can check whether the IDE launched our application, or whether it probably did not, by identifying if the integrated debugger is in control of our application. If it is, then Delphi (or some suitable replacement debugger) launched us.

An alternative approach to this problem, that could be considered more reliable, would be based upon this fact: if Delphi launches an application, then whilst the application is executing Delphi still has the project loaded. A side effect of Delphi having a project loaded is that the project title is written in the caption bar on Delphi's main window. Listing 7 has code that locates Delphi's main window, reads the caption and looks for the app's name.

The catch to this routine is that if Delphi launches an application when integrated debugging is disabled, there is nothing stopping the user loading a new project: Delphi will not complain. As well as this, if the user has several Delphi sessions active, this code will simply find the first Delphi main window, which may not be the correct one. However, you may agree with me that these disadvantages are reasonably minor.

The code from all these listings is in the project IDERun.Dpr on the disk, to save you typing it in. Figure 1 shows what it looks like after having been launched from Delphi 4, with integrated debugging disabled, on a machine with Delphi 1 also installed.

Thanks to Søren Jensen for comments used in this article.

Brian Long is an independent consultant and trainer. You can reach him at brian@blong.com

➤ *Above: Listing 5*

```
function DelphiDebuggerRunning: Boolean;
begin
  Result := Bool(PrefixSeg) and Bool(PWordArray(MemL[DSeg:36])^[8])
end;
```

➤ *Below: Listing 6*

```
function DelphiDebuggerRunning: Boolean;
begin
  Result := DebugHook <> 0;
end;
```

➤ *Listing 7*

```
function DelphiLaunchedMe: Boolean;
var
  Wnd: HWnd;
  CCaption: array[0..255] of Char;
  FileName, Caption: String;
begin
  Result := False;
  if DelphiRunning then begin
    { Get Delphi's main window }
    Wnd := FindWindow('TAppBuilder', nil);
    { Read its caption }
    GetWindowText(Wnd, CCaption, SizeOf(CCaption));
    { Translate the C string into a Pascal string, upper cased }
    Caption := UpperCase(StrPas(CCaption));
    { Find the root part of this project name... }
    FileName := ExtractFileName(Application.ExeName);
    { ...without the extension }
    FileName := Copy(FileName, 1, Length(FileName) - 4);
    { If Delphi has my project name in its caption, then we win }
    Result := Pos(FileName, Caption) <> 0;
  end
end;
```

➤ *Figure 1*