The Delphi CLINIC

Edited by Brian Long

Problems with your Delphi project?

Just email Brian Long, our Delphi Clinic Editor, on clinic@blong.com

Conditional Symbols

Q I have a huge application with several units. Some of them are to be shared among different applications. In those shared units, I want to compile only certain parts of the unit for some applications. This way, I would be able to restrict functionality with ease.

I've tried it with conditional defines but it seems that `{$DEFINE name}` only works for the unit where it is defined. Is that so?

Firstly, I placed the define symbol `{$DEFINE SomeName}` in the main form, and then tried to move it to the .dpr. None worked. Finally, I moved it to the unit itself and it did not work either.

In the units where I want to restrict compilation, I placed:

```
{$IFDEF SomeName}
...code...
{$ENDIF}
```

The result is that this works when omitting the parts where I have the `{$IFDEF....}` condition, but that's exactly the opposite behaviour to what I am after. Am I missing something?

A As you spotted, `$DEFINE` works from the point it occurs downwards in that unit only. Clearly, if you take that at face value, you get the impression that conditional compilation isn't that useful. However, you can define a symbol globally by using the `Directories/Conditionals` tab in the Project Options dialog, which will be stored in the project's DOF (Delphi) or KOF (Kylix) project options file.

Alternatively, you can place all the defines in an include file (.INC) and include it with the `$I` or `$INCLUDE` directive at the top of *every* unit (well, every unit that will potentially be affected anyway).

RTTI And Interfaces

Q Does Delphi RTTI support interfaces yet? I have a component that has a published interface property which is implemented by another component and need the implementing component streamed out in lieu of the interface property. But it doesn't work.

A The source code to the TypInfo.pas unit would give the true answer to the question, but it seems that as of Delphi 5 the answer is 'no'. However, this is set to change. Following are some excerpts from the transcript of an online chat from some time during last year that I found on Borland's website. The responses are from members of Delphi R&D:

matra: *What is the best way to stream out such properties (we should probably stream out reference to* `TComponent` *which implements the interface)?*

dthorpe: *Currently, you can implement your own streaming of interface references by defining your own 'give me your implementing component' interface and streaming out the component reference as normal. We are investigating ways to provide this support throughout the VCL. Stay tuned.*

jons_71: *Could properties of type Interface be placed in Object Inspector?*

RobertKozak: *There is ongoing development on interfaces. Stay tuned.*

tgrubb: *Are there plans to support properties whose type is an interface at design time? ie, I have a component that implements* `IWidget` *and another component that has a property of type* `IWidget`*. I would like to be able to assign the property at design time.*

dthorpe: *Development on interfaces is ongoing. Stay tuned.*

To me, this suggests that Delphi 6 will almost definitely support RTTI on interface properties and, given that Kylix already does, this makes it a dead cert.

On this month's disk is an example of streaming a component with an interface property that works fine in Kylix. The IntfPropTest.pas unit defines two component classes (`TIntfPropTest` and `TIntfImplementor`) and an interface type (`IMyInterface`) as shown, `TIntfPropTest` defines a property of type `IMyInterface` and `TIntfImplementor` implements the `IMyInterface` interface.

➤ *Listing 1: A streamable interface property.*

```
type
  IMyInterface = interface(IInterface)
  ['{329F5DD8-697B-46E2-9ABD-07BD7F439C73}']
    procedure DoSomething;
  end;
  TIntfImplementor = class(TComponent, IMyInterface)
  protected
    procedure DoSomething;
  end;
  TIntfPropTest = class(TComponent)
  private
    FIntfProp: IMyInterface;
  protected
    procedure SetIntfProp(Value: IMyInterface);
  public
    procedure Notification(AComponent: TComponent;
      Operation: TOperation); override;
  published
    property IntfProp: IMyInterface read FIntfProp write SetIntfProp;
  end;
```

In order to work smoothly with the streaming system, there are a couple of requirements you must meet. Firstly, the interface property must have a setter method (called when the property is assigned) which calls the component's `ReferenceInterface` method both before and after storing the new interface value, as shown in Listing 2.

According to comments in the `Classes` unit, `ReferenceInterface` establishes (`opInsert`) or removes (`opRemove`) internal links that notify the calling component when the component that implements the given interface is destroyed. The function result indicates whether the function was able to establish/remove a notification link or not.

A result of `False` doesn't necessarily indicate an error, but it does mean that the interface's implementer does not participate in the interfaced component reference model. This could mean that the given interface employs true reference counting, independent of component lifetimes. That doesn't affect the use of interface properties at runtime, but non-component interfaces cannot be stored by the property streaming system. If you are confident that your interface is component-based, then you can ignore the return value.

The next requirement is to override the `Notification` method so you can tell if the component implementing your interface is destroyed (see Listing 3). Notice that `nil` is being assigned to the property, not the private data field. This ensures that the `ReferenceInterface` methods are called correctly. All assignments to the interface property *must* be made through the property setter.

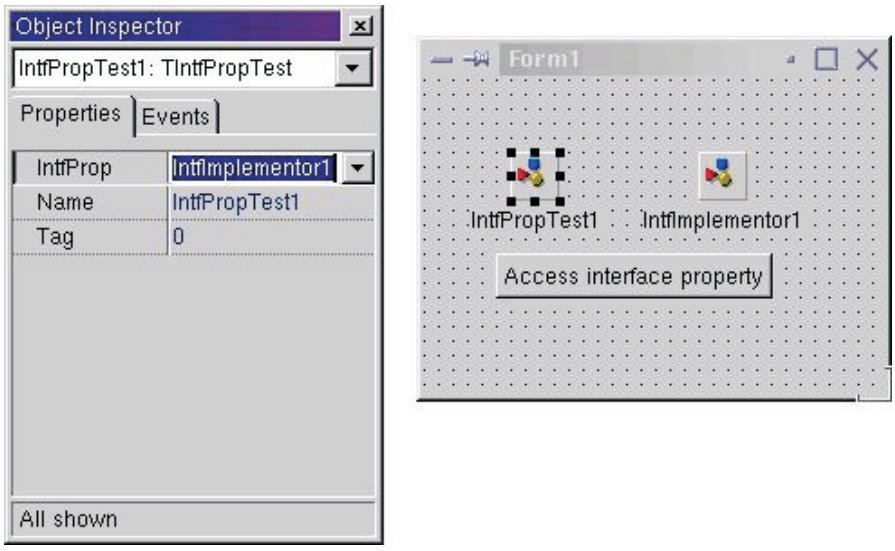
Also on the disk is a simple CLX project that has an instance of both components on the form (see Figure 1). The `TIntfPropTest` interface property has been linked to the interface implementing component and a button calls the interface method `DoSomething` through the interface property (shown in Listing 4). The project runs in Kylix and hopefully will also work in

```
procedure TIntfPropTest.SetIntfProp(Value: IMyInterface);
begin
  ReferenceInterface(FIntfProp, opRemove);
  FIntfProp := Value;
  ReferenceInterface(FIntfProp, opInsert);
end;
```

➤ *Listing 2: The interface property setter.*

```
procedure TIntfPropTest.Notification(AComponent: TComponent;
  Operation: TOperation);
begin
  inherited;
  if Assigned(IntfProp) and AComponent.IsImplementorOf(IntfProp) then
    IntfProp := nil;
end;
```

➤ *Listing 3: Detecting the last moments of the implementing component.*



➤ *Figure 1: The interface property in the Object Inspector.*

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  IntfPropTest1.IntfProp.DoSomething
end;
```

➤ *Listing 4: Calling an interface method through the interface property.*

```
procedure TPicture.SetGraphic(Value: TGraphic);
var
  NewGraphic: TGraphic;
begin
  NewGraphic := nil;
  if Value <> nil then begin
    NewGraphic := TGraphicClass(Value.ClassType).Create;
    NewGraphic.Assign(Value);
    NewGraphic.OnChange := Changed;
    NewGraphic.OnProgress := Progress;
  end;
  try
    FGraphic.Free;
    FGraphic := NewGraphic;
    Changed(Self);
  except
    NewGraphic.Free;
    raise;
  end;
end;
```

➤ *Listing 5: The VCL code for assigning a graphic object to a picture object.*

Delphi 6 when it ships, as Delphi 6 will also support the CLX library.

Graphics Query

Q I have an application that uses a `TImage` component to display a variety of bitmaps, and it seems to have a memory leak. I have followed the instructions in the online help which says that the image component will take ownership of the bitmap when assigned to it and consequently I cannot see where the problem is. Is there an inherent problem with using `TImage` components?

A Having looked through the help, I can see where you got your information from. From the Delphi help contents page, choose *Programming with Delphi*, then *Working with graphics & multimedia*, then choose either *Setting the initial bitmap size* or *Replacing the picture*.

Depending which topic you go to, you will find one of these two statements:

Assigning the bitmap to the picture's `Graphic` *property gives ownership of the bitmap to the picture object. The picture object destroys the bitmap when it finishes with it, so you should not destroy the bitmap object. You can assign a different bitmap to the picture at which point the picture disposes of the old bitmap and assumes ownership of the new one.*

Note: assigning a new bitmap to the picture object's `Graphic` *property causes the picture object to destroy the existing bitmap and take ownership of the new one. The VCL handles the details of freeing the resources associated with the previous bitmap automatically.*

As mentioned, these categorically state that the `Picture` object in a `TImage` will take care of destroying any bitmap you assign to the `Graphic` property. Unfortunately these are both incorrect. When something is assigned to the `Graphic` property of a `TPicture`, the code in Listing 5 is executed.

`FGraphic` is the old graphic object (which will probably be a `TBitmap`, `TIcon`, or `TMetafile`). Notice that it creates a new graphic object of a matching type, copies the contents of your graphic object across, and frees the old copied graphic object. If the value assigned to the property is `nil`, no new object is created, the old one is just destroyed.

But the key point here is that nowhere does the code execute anything like:

```
FGraphic := Value
```

which would be necessary for it to take responsibility for freeing your object.

This problem has been reported before, and it seems that Borland are responding to it, as the equivalent Kylix help topics show correct information. The two equivalent Kylix help statements are:

Assigning the bitmap to the picture's `Graphic` *property copies the bitmap to the picture object. However, the picture object does not take ownership of the bitmap, so after making the assignment, you must free it.*

Note: Assigning a new bitmap to the picture object's `Graphic` *property causes the picture object to copy the new graphic, but it does not take ownership of it. The picture object maintains its own internal graphic object.*

I expect the Delphi 6 help to also have these corrections.

Intercepting Window Closure

Q When I show my login screen for my program, how can I prevent the user pressing `Alt+F4` to close the dialog? I want to keep it open.

A The `Alt+F4` key sequence causes Windows to send a `WM_CLOSE` message to the window. This is an instructional message and the window can refuse to play ball. The Delphi way of rejecting a close instruction is to write an `OnCloseQuery` event handler for the login form and set the `CanClose` parameter to `False`. Alternatively, you could write an `OnClose` event handler and set the `Action` parameter to `caNone`.

Abbreviated Emailing

Q I can easily issue a call such as the following:

```
ShellExecute(handle, 'open',
  'mailto:user@company.com',
  nil, nil, SW_SHOWNORMAL);
```

to start my Outlook with the `To...` field filled in. What I don't know how to do is to add text into the subject field or into the contents or how to add an attachment. Do you know this?

A To begin with, this sounds like you are looking for the definition of a subset of the URL syntax. The full URL syntax is dictated by Internet RFC 1738 (which is at ftp://ftp.isi.edu/in-notes/rfc1738.txt). The `mailto` URL scheme is described at length in RFC 822 (ftp://ftp.isi.edu/in-notes/rfc822.txt).

However, having browsed these documents, they didn't seem to be telling me what I expected, so I did some searching on MSDN. The result was that you can execute a URL such as the one in Listing 6 and a message will be started in Outlook that has the `To...` field, `Cc...` field, `Subject` line and also some body text set up.

Note that you must translate various characters (high ASCII characters, spaces and non-printable characters) into 3-character sequences starting with % and followed by a two-digit hexadecimal representation of the ASCII code. This explains the occurrence of `%20` (space), `%0D` (carriage return) and `%0A` (line feed) in the example shown.

I have not found a way of specifying an attachment so far.

By the way, Internet Explorer defines a maximum URL length of

➤ *Listing 6: A URL that will set up a mail message.*

```
mailto:user@company.com?CC=otheruser@company.com?Subject=Interesting%20
  topic&Body=hello%20world%0D%0Ahello%20again
```

2,083 characters with a maximum path length of 2,048 characters.

Custom App Launching

Q I run other executables from within a base application using:

```
Win32Check(CreateProcess(nil,
  ProgramToRun,nil,nil,False,
  0,nil,nil,SI,PI))
```

With the `WinExec` function on Win 3.1 I could use `SW_SHOWMAXIMIZED` as the second parameter to force the application to maximise its main window. Is there any way I can do this with the above method?

A When calling `CreateProcess`, the second from last parameter is an input record that you should set up in advance to describe various startup attributes for the application being launched.

If you are not interested in specifying any values other than the `ShowWindow` flag you mentioned, then you can ask Windows to duplicate the startup details for your base application and then simply set the `ShowWindow` flag afterwards. So you could precede the `CreateProcess` call with the statements in Listing 7.

You can find out more about `CreateProcess` by looking at *The Delphi Clinic* in past issues of *The Delphi Magazine*. For example, *CreateProcess Alert* in Issue 51, *Terminating Programs* in Issue 49, *Capturing DOS Output* in Issue 37, *Application & Window Handles* in Issue 32, *Are You Running?* in Issue 28, *Hi MOM!* from Issue 23 and *Waiting For Termination* in Issue 16.

TOleContainer And In-place Editing

Q I am using the `TOleContainer` class to embed an 'Excel.Sheet.8' object within a Delphi app container. The main line is:

```
Container1.CreateObject(
  'Excel.Sheet.8', False );
```

Here is my problem. When Excel is not running on the system, the line above starts a new copy of Excel, which is fine. However, if it is already running, the line above uses the existing Excel process (actually the first launched Excel process in case of many processes).

I'd like the container to have its own process in all cases. Is that possible?

I looked into the `TOleContainer` code and observed that the `OleCreate` API was called, but couldn't find a flag or parameter for this routine that governs whether a new instance is started or whether it tries to find a running instance first.

A When I was looking into this problem I checked what the `TOleContainer.CreateObject` method did, and it uses a `TCreateInfo` record which has a `CreateType` field. `CreateObject` sets this to have a value of `ctNewObject`, which, as you say, causes `OleCreate` to be called.

When creating your own COM objects, you can either call `GetActiveObject` to locate an already running instance of a server (as registered in the ROT), or you can call `CoCreateInstance` to create a new object. It would appear that `OleCreate` tries `GetActiveObject` first and, if this fails, it then calls `CoCreateInstance`.

As you say, there is no flag to control whether or not it will do this, so it looks like you are stuck with the way it works, unless any knowledgeable readers can show otherwise: drop me an email if so!

```
GetStartupInfo(SI);                 //get startup info from this current process
SI.wShowWindow := SW_MAXIMIZE;      //Make new app start maximised
```

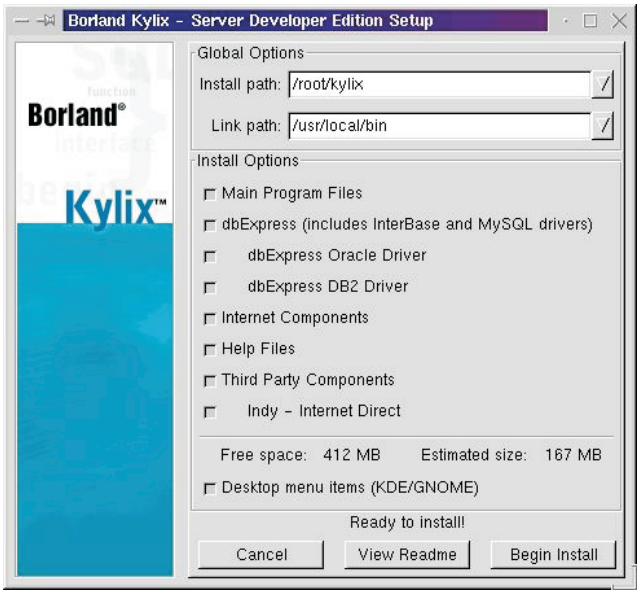
➤ *Listing 7: Specifying how an application should display its main window.*

Installing Kylix

Q Before I installed Kylix I checked the README file and it suggested I log in as `root` to install it. I duly did this, and the installation program suggested a path of `/root/kylix` which I accepted (see Figure 2). The trouble is that only `root` has access to that directory and so I can only run Kylix when logged in as `root`. What should I have done?

A The suggestion of installing as `root` is good, as this *should* enable all users to run Kylix after installation. By default the installation program suggests a subdirectory under the logged in user's home directory, hence the suggestion of `/root/kylix`. However, as you have spotted, in the case of `root`, this home directory is very unhelpful as only `root` has access to it.

What you should do is choose one of the alternative installation directories available from the combobox where `/root/kylix` was offered (see Figure 3). In addition to installing Kylix under the `/root` directory, you can choose to install it under `/usr/local`, `/usr` or in `/opt`. You can also type in



➤ *Figure 2: The installation program offering an unsuitable installation path.*

an entirely different path if you wish to.

I understand JBuilder for Linux defaults to installing in the `/opt` directory, but it is also very common for custom software to be installed somewhere in the `/usr` directory tree.

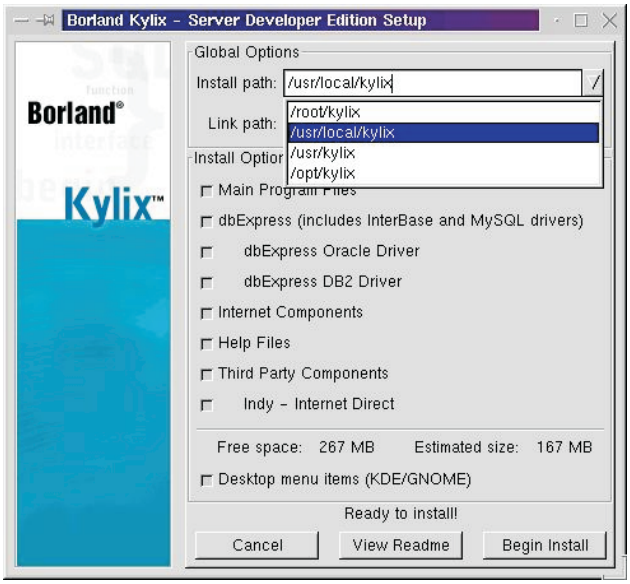
By the way, the Kylix INSTALL file does have something to say on the matter of installing Kylix as the `root` user: *the default installation location for Kylix is always beneath the home directory of the installing user. But Kylix should never be installed in* `/root`. *That directory is not normally accessible to other users, and running Kylix as* `root` *is not recommended. When installing Kylix as* `root`, *always specify another installation location, such as* `/usr/local/kylix`.

Kylix Configuration Files

Q I haven't got a copy of Kylix yet but was wondering how it saves state information between sessions, what with there not being a Windows registry on Linux.

A Assuming Kylix has been installed by user `root`, as described above, then you should see the files in Table 1 created. These are default configuration files, used when the Kylix user does not have any local override files.

The WINE file might warrant some explanation. The Kylix IDE carries much of its old Windows-specific code inside. To avoid rewriting the whole thing from scratch in Linux with CLX, Borland chose to compile parts of the WINE Windows emulation library into the IDE to allow the Windows/VCL code to compile into a native Linux ELF (**E**xecutable and **L**inking **F**ormat) executable. This meant the IDE could be made available in a much shorter time than if it was completely re-written in CLX. I gather they are planning to have a true Linux IDE ready for version 2.



➤ *Figure 3: Choosing a better installation path.*

The files in Table 1 can be considered to contain the global options. When a user first runs Kylix, a .borland directory is made in their home directory (~/.borland). This is a hidden directory, thanks to the first character being a full stop, but you can see hidden files at the prompt by running:

```
ls -a
```

In this directory are local configuration files which, in many cases, override the global defaults (see Table 2). It would appear that when Kylix starts, it checks whether the ~/.borland directory exists. If it doesn't, it creates it and copies over the original configuration files so the user's specific options do not interfere with anyone else's.

The first time you run Kylix you will see a dialog indicating that some font metrics are being saved. These are calculated for WINE's benefit and are stored in ~/.borland/cachedmetrics:0.

Kylix And Linux Distros

Q I run a Linux distro other than the ones officially supported by Kylix. Does this mean I will not be able to run Kylix on my distro?

A The launch materials for Kylix 1.0 advertise Red Hat 6.2 and later, SuSE 7.0 and later and Mandrake 7.2 or later as the officially supported Linux distributions (or *distros*). In fact Borland even ship a copy of SuSE 7.0 in the Kylix box.

➤ *Table 1: Global Kylix configuration files.*

Configuration File	Purpose
`/usr/local/etc/delphi60rc.conf`	General Kylix information, such as root installation directory, IDE packages, debug DCU path, help file location, browsing paths
`/usr/local/etc/delphi60dci.conf`	Default code templates invoked with Ctrl+J
`/usr/local/etc/delphi60dmt.conf`	Default menu templates used in the Menu Designer
`/usr/local/etc/delphi60dro.conf`	Object Repository setup
`/usr/local/etc/dcc.conf`	Default command-line compiler options
`/usr/local/etc/dbxconnections.conf`	dbExpress connection configuration
`/usr/local/etc/dbxdrivers.conf`	dbExpress driver configuration
`/usr/local/etc/borlandrc.conf`	WINE configuration file

However, this list simply covers the distributions that have been thoroughly tested by Borland. With the current certified distros, Borland supply patches on the Kylix CD that ensure any problems that might be present are eliminated.

Each different distribution vendor ships whatever version of the Linux kernel they choose, along with a whole gamut of libraries and applications. If Kylix doesn't work on a given distribution, it will be because various libraries are too old or not present. You can always download the appropriate up-to-date versions of the key libraries used by Kylix and upgrade your distribution to a suitable level of compatibility.

As time goes by, Borland will test Kylix against further distributions (whose vendors all appear keen to meet the requirements so Kylix developers might use their distro) and certify them. Until then, Kylix *can* be made to run on other distros, but Borland will not support those configurations until Kylix is certified on them. In fact the Kylix README discusses how to overcome some font issues in the Chinese version of TurboLinux.

Other than a certified distro, the current Kylix requirements are as follows.

Software requirements:

- Linux kernel version 2.2 or higher. You can find your kernel release number by running `uname` with the `-r` switch.
- glibc 2.1.2 or later. Kylix applications also require this.
- libgtk.so version 1.2 or higher. This is required only by the graphical installation program to run. If it is not present, text mode prompts will be used instead.
- libjpeg version 6.2 (that is, libjpeg.so.62) or higher.
- An X11R6-compatible terminal server, such as XFree86, for the Kylix GUI.

Hardware requirements:

- Intel Pentium 200 MHz (Pentium II 400 MHz recommended).
- 64Mb RAM (128Mb is recommended).
- CD-ROM drive.
- 175Mb hard disk space for a full installation.
- VGA or higher resolution monitor.
- Mouse or other pointing device.

Configuration File	Purpose
~/.borland/delphi60rc	User-specific version of delphi60rc.conf
~/.borland/delphi60dci	User-specific code templates
~/.borland/delphi60dmt	User-specific menu templates
~/.borland/delphi60dro	User-specific Object Repository setup
~/.borland/delphi.dct	User's component templates
~/.borland/defproj.conf	User-specific default command-line compiler options
~/.borland/defproj.kof	User-specific default IDE project options
~/.borland/dbxconnections	User-specific dbExpress connection configuration
~/.borland/dbxdrivers	User-specific dbExpress driver configuration
~/.borland/cachedmetrics:0	WINE font metrics
~/.borland/borl~wex.con	User-specific version of borlandrc.conf
~/.borland/wineserver-*hostname*:0	Directory for WINE usage

➤ *Table 2: User-specific Kylix configuration files.*

➤ *Figure 4: The new Easter Egg: kudos!*

Kylix Easter Eggs

Q Does Kylix have any new Easter Eggs I should look out for?

A In the About box, you can still hold down the `Alt` key whilst typing in appropriate words to get a scrolling list of names. For example, the old `Alt+TEAM`, `Alt+QUALITY` and `Alt+DEVELOPERS` still work fine, as does `Alt+JEDI`. They have added in one new one that I have found so far, which is `Alt+KUDOS`. This gives a scrolling list of 'Thank Yous' to people at CodeWeavers (for the work on WINE), the people behind the cross-platform Indy Internet components and also to Xapware (see Figure 4).

By the way, you can always find out about the Borland Easter Eggs I have bumped into by checking my website at www.blong.com/Undocumented/EasterEggs.htm.