

Kylix Language Enhancements

by Bob Swart

Apart from offering RAD development on Linux, I have just discovered a few minor changes in the Kylix compiler, which may or may not please you, and which may or may not be made available in the upcoming Delphi 6. More about Delphi 6 next time when I'll have access to Delphi 6 itself to compare.

Constant Typed Constants

Delphi 5 introduced the so-called initialised variables, where a variable declaration could immediately be followed by a value to initialise the variable, for example:

```
var
  PerInitialisedVariable: Integer = 42;
```

Although the syntax was new to us, the concept was not. In previous versions of Delphi we could already use something called typed constants. A typed constant is a constant definition with a typename included (almost like an initialised variable), for example:

```
const
  ConstConstant = 42; // cannot change value
  TypedConstant: Integer = 7;
```

I still don't know if it's a bug or a feature, but typed constants in Delphi can be changed as well (i.e. they are in fact not truly constant), so we can write the following line of code to change the value of the type constant TypedConstant to 42.

```
begin
  TypeConstant := 42
end.
```

The fact that typed constants were not in fact constant had a significant effect on the use of these values. Real constant constants can be used as case labels, for example, but typed constants cannot. See the following code for example:

```
case SomeInteger of
  ConstConstant: writeln('Yes');
// TypedConstant: writeln('No Compile');
  else writeln('No')
end;
```

Note that typed constants cannot be used as case labels since the compiler is unable to determine (at compile time) the value of the typed constant.

Apart from global typed constants, you could also define a typed constant inside a routine, resulting in a local typed constant. Again, the typed constant can be modified inside this routine (and only inside the routine, since nobody outside the routine can see it). But as a special side-effect of typed constants, it will be initialised only once, and at all other times retain its current value. So, if you start with a value of 1, and modify it inside the routine then, next time you enter the routine, the modified value will be there (waiting for you). You can use this feature as a simple counter, to measure the amount of times a routine is called:

```
procedure Routine;
const
  CounterOnlyVisibleInsideRoutine: Integer = 0;
begin
  Inc(CounterOnlyVisibleInsideRoutine);
end;
```

The local typed constant *CounterOnlyVisibleInsideRoutine* can be seen as a global variable with local scope (only inside the Routine) and full lifetime. I love them because, apart from counters, you can also use them to implement a simple history feature. And since their scope is local, you are assured that nothing else can touch them or modify them.

Save and clean global variables. *What else do you want?* Needless to say, I use them a lot.

Enter Kylix

For some reason (and I hope this is a temporary problem, and fixed in the next version), Kylix considers typed constants real constants. You cannot change them. In fact, statements that attempt to do so will result in a genuine compiler error! This breaks a lot of legacy code for me. I have no choice but to convert the clean local typed constant into a truly global (initialised) variable or global typed constant. This resulting in code which is less clean and in my view a bit harder to maintain.

```
const
  CounterNowAlsoVisibleOutsideRoutine: Integer
    = 0;

procedure Routine;
begin
  Inc(CounterNowAlsoVisibleOutsideRoutine);
end;
```

Again, I hope they fix this in the next release of Kylix (and I sure hope they don't enforce the same constant type constants rule in Delphi 6).

Enumerated Enumerations

There are a few areas in which the C/C++ syntax is not only less readable but also - let's be honest - more powerful than Delphi's ObjectPascal syntax (one could argue that the latter often results in the former). One of the features available in C/C++ is the ability to assign specific values to enumeration types. This is not possible in Delphi 5. But it is possible in Kylix (so hopefully in Delphi 6 as well). We can now write the following:

```
type
  QMessageBoxButton = (NoButton = 0,
    Ok = 1,
    Cancel = 2,
    Yes = 3,
    No = 4,
    Abort = 5,
    Retry = 6,
    Ignore = 7,
    ButtonMask = $07,
    Default = $100,
    Escape = $200,
    FlagMask = $300);
```



Note that the value of ButtonMask is the same as Ignore, and that there are significant gaps between ButtonMask and Default, Escape and FlagMask.

This certainly is a neat feature, which won't compile but may have a good chance of working inside Delphi 6 (more on this topic next time, of course).

Linux Harddisk Performance

A final - totally unrelated - tip is regarding Linux hard disk performance, which wasn't very fast on my old machine. To get an indication of the read speed of your hard disk under Linux, type:

```
hdparm -t /dev/hda
```

On my machine, this returns 32 MB in 10.91 seconds or 2.93 MB/sec. Not very fast or impressive. In order to speed this up a little bit, you could type:

```
hdparm -c1 -d1 /dev/hda
```

to enable 32-bit I/O support and turn on the use of DMA. As a result, on my machine I now have 32 MB in 5.60 seconds or 5.71 MB/sec. The disk reads almost twice as fast now!



Bob Swart (aka *Dr. Bob* - www.drbob42.com) is an @-Consultant for Everest Delphi OplossingsCentrum and has spoken at conferences all over the world. He's also a free-lance technical author and has written chapters for several Delphi and C++Builder books.