



Marco Cantu 8 Nov 2018

The coming 10.3 version of Delphi introduces a very handy feature to the language, local inline variables with local scope and type inference.

The Delphi language in 10.3 has a fairly core change in the way it allows far more flexibility in the declaration of local variables, their scope and lifetime. This is a change that breaks a key tenet of the original Pascal language, but offers a significant number of advantages, reducing unneeded code in several cases.

Old Style Var Blocks

Since Turbo Pascal 1 and until now, following classic Pascal language rules, all local variable declarations had to be done in a var block written before the beginning of a function, procedure or method:

```
procedure Test;
var
  I: Integer;
begin
  I := 22;
  ShowMessage (I.ToString);
end;
```

Inline Variable Declarations

Recommended

["big" bug in c++ builder 10..2 - math.h](#)

[bug viewing UnicodeString value in IDE debugger](#)

[EKON 23 in October and UK RoadShow in November](#)

Related

[Directions for ARC Memory Management in Delphi](#)

[Yet another inline variables bug](#)

[My Delphi software is empty and I can't](#)

The new inline variable declaration syntax allows you to declare the variable directly in a code block (allowing also multiple symbols as usual):

```
procedure Test;
begin
  var I, J: Integer;
  I := 22;
  j := I + 20;
  ShowMessage (J.ToString);
end;
```

While this might seem a limited difference, there are several side effects of this change. It is these additional effects that make the feature valuable.

Initializing Inline Variables

A significant change compared to the old model, is that the inline declaration and initialization of a variable can be done in a single statement. This makes things more readable and smoother compared to initializing several variables at the beginning of a function.

```
procedure Test; // declaration and initialization in a single statement
begin
  var I: Integer := 22;
  ShowMessage (I.ToString);
end;
```

More over, if the value of a variable is available only later in the code block, rather than setting an initial value (like 0 or nil) and later assign the actual value, you can delay the variable declaration to the point you can calculate a good initial value:

```
procedure Test1; // multiple inline declarations (symbols declared later)
begin
  var I: Integer := 22;
  var J: Integer := 22 + I;
  var K: Integer := I + J;
  ShowMessage (K.ToString);
end;
```

In other words, while in the past all local variables were visible in the

Recommended

["big" bug in c++ builder 10..2 - math.h](#)

[bug viewing UnicodeString value in IDE debugger](#)

[EKON 23 in October and UK RoadShow in November](#)

Related

[Directions for ARC Memory Management in Delphi](#)

[Yet another inline variables bug](#)

[My Delphi software is empty and I can't](#)

Scope and Lifetime of Inline Variables Declared in Nested Blocks

The scope limitation is also more relevant as it doesn't apply to the entire procedure or method, but only to the begin-end code block the inline variable appears. In other words, the scope of an inline variable is limited to the declaration block and the variable cannot be used outside of the block. Not only, but the variable lifetime is also limited to the block. A managed data type (like an interface, string or managed record) will now get disposed at the end of the sub-block, rather than invariably at the end of the procedure or method.

```
procedure Test2; // scope limited to local block
begin
  var I: Integer := 22;
  if I > 10 then
  begin
    var J: Integer := 3;
    ShowMessage (J.ToString);
  end
  else
  begin
    var K: Integer := 3;
    ShowMessage (J.ToString); // COMPILER ERROR: "Undeclared identifier: J"
  end;
  // J and K not accessible here
end;
```

As you can see in the last code snippet above, a variable declared inside a begin-end block is visible only in the specific block, and not after the block has terminated. At the end of the if statements, J and K won't be visible any more.

As I mentioned, the effect is not limited only to visibility. A managed variable, like an interface reference or a record, will be properly cleaned up at the end of the block, rather than at the end of the procedure or method:

```
procedure Test99;
begin

  // some code

  if (something) then
  begin
    var Intf: IInterface = GetInterface; // Intf.AddRef
    var MRec: TManagedRecord = GetMRecValue; // MRec.Create + MR
    UseIntf(Intf);
    UseMRec(MRec);
  end; // Intf.Release and MRec.Destroy are implicitly called at
```

Recommended



["big" bug in c++ builder 10..2 - math.h](#)

[bug viewing UnicodeString value in IDE debugger](#)

[EKON 23 in October and UK RoadShow in November](#)

Related

[Directions for ARC Memory Management in Delphi](#)

[Yet another inline variables bug](#)

[My Delphi software is empty and I can't](#)

```
end; // no additional cleanup
```

Type Inference for Inline Variables

Another huge benefit of inline variables is that the compiler now can, in several circumstances, infer the type of an inline variable by looking to the type of the expression or value assigned to it:

```
procedure Test;
begin
  var I := 22;
  ShowMessage (I.ToString);
end;
```

The type of the r-value expression (that is, what comes after the `:=`) is analyzed to determine the type of the variable. Some of the data types are “expanded” to a larger type, as in the case above where the numeric value 22 (a *ShortInt*) is expanded to *Integer*. As a general rule, if the right hand expression type is an integral type and smaller than 32 bits, the variable will be declared as a 32-bit Integer. You can use an explicit type if you want a specific, smaller, numeric type.

Now while this feature can save you a few keystrokes for an Integer or a string, variable type inference becomes fairly nice in case of complex type, like instances of generic types. In the code snippet below, the types inferred are *TDictionary* for the variable *MyDictionary* and *TPair* for the variable *APair*.

```
procedure NewTest;
begin
  var MyDictionary := TDictionary (string, Integer).Create
  MyDictionary.Add ('one', 1);
  var APair := MyDictionary.ExtractPair('one');
  ShowMessage (APair.Value.ToString)
end;
```

(Code above uses parenthesis rather than angle brackets for the generic types to avoid the issues on this blog, sorry)

Inline Constants

Beside variables, you can now also inline a constant value declaration

Recommended

["big" bug in c++ builder 10..2 - math.h](#)

[bug viewing UnicodeString value in IDE debugger](#)

[EKON 23 in October and UK RoadShow in November](#)

Related

[Directions for ARC Memory Management in Delphi](#)

[Yet another inline variables bug](#)

[My Delphi software is empty and I can't](#)

```
const M: Integer = (L + H) div 2; // single identifier, with type specifier
const M = (L + H) div 2; // single identifier, without type specifier
```

For Loops With Inline Loop Variable Declaration

Another specific circumstance in which you can take advantage of inline variable declarations is with for loop statements, including the classic for-to loops and modern for-in loops:

```
for var I: Integer := 1 to 10 do ...
for var Item: TItemType in Collection do...
```

You can further simplify the code taking advantage of type inference:

```
for var I := 1 to 10 do ...
for var Item in Collection do ...
```

This is a case in which having the inline variable with limited scope is particularly beneficial, as in the sample code below: Using the *I* variable outside of the loop will cause a compiler error (while it was only a warning in most cases in the past):

```
procedure ForTest;
begin
  var total := 0;
  for var I: Integer := 1 to 10 do
    Inc (Total, I);
  ShowMessage (total.ToString);
  ShowMessage (I.ToString); // compiler error: Undeclared Identifier
end;
```

Inline Summary

Inline variable declarations, with type inference and local scope, bring a new level of readability to Pascal source code. While Pascal source code readability remains a key tenet to preserve the language at the core level, removing some of the rust (real or perceived) are so many advantages to depart from the traditional coding style.

Recommended

["big" bug in c++ builder 10..2 - math.h](#)

[bug viewing UnicodeString value in IDE debugger](#)

[EKON 23 in October and UK RoadShow in November](#)

Related

[Directions for ARC Memory Management in Delphi](#)

[Yet another inline variables bug](#)

[My Delphi software is empty and I can't](#)

Still, you can keep your code as is and ask your fellow developers to stick with the traditional declaration: nothing in the existing code or style is wrong, but inline declarations offer you a new opportunity. I already have trouble going back to older versions of Delphi... just saying.

3 comments 1 member is here

Top Comments



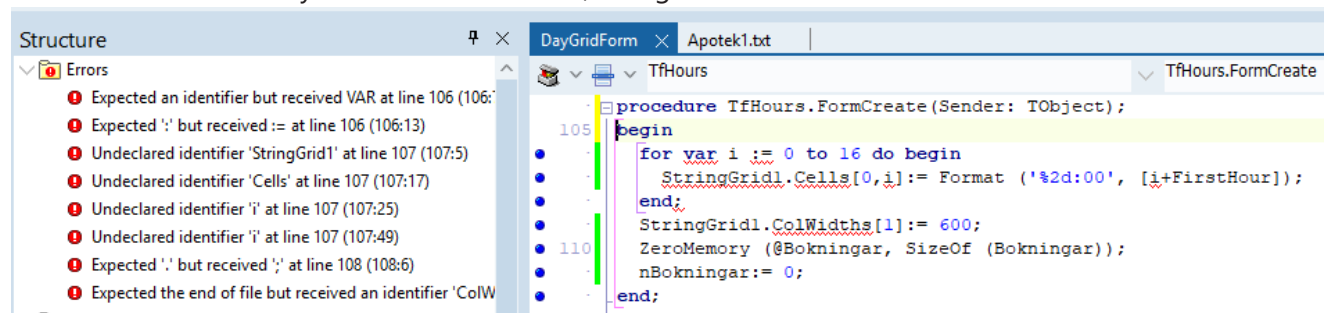
Offline [kenrun](#) 10 months ago +4

Very nice feature! I like it a lot and use it a lot. BUT - I hope you will fix this one annoying bug soon: for the v...



Offline [kenrun](#) 10 months ago

Very nice feature! I like it a lot and use it a lot. BUT - I hope you will fix this one annoying bug soon: for the very first procedure/function in a file in which you use inline variables, the IDE flags the code as erroneous with red wavy lines. The code works, though.



[Didier D9129](#) 9 months ago in reply to [kenrun](#)

Not only this bug is very annoying, but at the same time, all functions and procedures which appear after the first inline declaration in the source disappear from the unrolling list in the editor window!



Offline [stefanya42](#) 10 months ago

This is a feature I'd been hoping for, though using it will make code backward incompatible quickly :-/

Recommended

["big" bug in c++ builder 10..2 - math.h](#)

[bug viewing UnicodeString value in IDE debugger](#)

[EKON 23 in October and UK RoadShow in November](#)

Related

[Directions for ARC Memory Management in Delphi](#)

[Yet another inline variables bug](#)

[My Delphi software is empty and I can't](#)