

Inline Variable Declaration

The Delphi language in 10.3 allows more flexibility in the declaration of local variables. Until now, following classic Pascal language rules, all 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;
```

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: Integer;  
  I := 22;  
  ShowMessage (I.ToString);  
end;  
  
procedure Test2;  
begin  
  var I, K: Integer;  
  I := 22;  
  K := I + 10;  
  ShowMessage (K.ToString);  
end;
```

While this might seem a limited difference, there are several side effects of this change. One is that declaration and initialization can be done in a single statement. A second side effect is that you can declare variable as needed in a complex code block, with limited scope (as the variable is visible only from the position of its declaration, and you don't need to have a variable declared and not initialized for portion of the code).

Scope of Inlined Variables

A third side effect is that the declaration is allowed also within a second level begin-end block and the scope is limited to that block.

Contents

Scope of Inlined Variables

Type Inference for Inlined Variables

Inline Constants

For Loops With Variable Declaration

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

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

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;
    end;
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.

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 + MRec.Assign
            UseIntf(Intf);
            UseMRec(MRec);
        end; // Intf.Release and MRec.Destroy are implicitly called at end of scope
```

```
// more code
```

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

Type Inference for Inlined Variables

Additionally, the compiler can now in several circumstances infer the type of a variable at its line declaration location, by looking to the type of the 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.

Also notice that only a single identifier can be declared without a value type (differently from general variable declarations and inline declarations).

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<string, Integer>`” for the variable `MyDictionary` and “`TPair<string, Integer>`” 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;
```

Inline Constants

Beside variables, you can now also inline a constant value declaration. This can be applied to types constants or untyped constants, in which case the type is inferred (a feature that has been available for constants for a long time). A simple example is below:

```
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 Variable Declaration

A specific circumstance in which you can take advantage of inline variable declarations is with loop statements, including for-to loops and 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 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 'I'
```

Retrieved from "http://docwiki.embarcadero.com/RADStudio/Rio/e/index.php?title=Inline_Variable_Declaration&oldid=268757"

This page was last edited on 21 January 2019, at 17:29.