Anonymous methods in Delphi.

Vasiliy V. Kevroletin

26 мая 2013 г.

Содержание

1 About

This document describes implementation of anonymous methods in Delphi. It tries to summarise things described in Delphi documentation and explore some not documented details using tests.

Here is Delphi documentation:
http://docwiki.embarcadero.com/RADStudio/XE3/en/Anonymous_Methods_in_Delphi

2 Notation

- A word “function” is the union of two related Pascal concepts: “function” and “procedure”.
- “Outer function” is an opposite to “nested function”. Example:

```
procedure Outer;
  procedure Nested; begin
  end;
begin
end;
```

- Note that Delphi have confusing terminology:
  - "Method pointer" is a pointer to method of object (object have named type).
  - "Method references" is a reference to anonymous method.

3 Differences between anonymous methods and functions

Few citation from Delphi documentation describes them:

- a procedure or function that **does not have a name**
- an anonymous method can refer to variables and bind values to the variables in the context in which the method is defined
- furthermore, these variables can be bound to values and wrapped up with a reference to the anonymous method. This captures state and **extends the lifetime of variables**.
- method references are managed types (they are reference counted)

4 Analogy between anonymous methods and functions

4.1 Nested functions can access local variables of outer function

Access to local variables of outer function from nested function happens through pointer to frame of outer function.

Example:

```
procedure Outer;
var a, b: Integer;
  procedure Nested;
  begin
    a := 10; b:= 20;
  end;
begin
  Nested();
end;
```

If compiler didn't support nested functions we would write:

```
type frame: record a, b: Integer; end;

procedure Nested(var f: frame);
begin
  f.a := 10; f.b:= 20;
end;

procedure Outer;
var f: frame;
begin
  Nested(f);
end;
```

4.2 Depth of nesting more than 2

```
procedure Sub1;
var a: Integer;
  procedure Sub2;
  var b: Integer;
    procedure Sub3;
    begin
      a := 10; b := 20;
    end;
  begin
    Sub3();
  end;
begin
  Sub2();
end;
```

If compiler didn't support nested functions we would write:

```
type frame1 = record
       a: Integer
     end;
     frame2 = record
       b: Integer;
       parent: ^frame1;
     end;

procedure Sub3(var parent: frame2);
begin
  parent.parent.a := 10;
  parent.b := 20;
end;

procedure Sub2(var parent: frame1);
var f: frame2;
begin
  f.parent := @frame1;
  Sub3(f);
end;

procedure Sub1;
var f: frame1;
begin
  Sub2(f);
end;
```

Idea is to make linked list of nested frames. It will allow to

- access any frame
- make recursive call of outer function (for example we can call Sub2 from Sub3 in example above)

4.3 Analogy between anonymous methods and object methods

1. An object is the data + functions which work with data.
2. An anonymous method is function + data which is used in function.

Someone can guess that there is a difference: (1) - is much data + many functions (2) - is much data + one function. So may be anonymous method is an object which have only one method. **This is not truth for Delphi**. In Delphi an anonymous method is the method of object associated with it's declaring routine. This object called FrameObject and it can have many methods if procedure associated with FrameObject have many anonymous methods. Such implementation have gotcha, it will be described below.

5 Summary about anonymous methods implementation in Delphi

1. All captured local variables become fields of special object associated with their declaring function. This object is called "FrameObject". It's allocated on heap and created once per function. FrameObject will be created iff

   - function have captured variables
   - function have anonymous method declared in it's body
   - in situation described in point 7.

   Access for captured variables happens through pointer to FrameObject.

   This is needed because captured variable can't be allocated on stack because their lifetimes can be longer than lifetime of their declaring function.

   Note that since captured values allocated on stack Delphi forbids capturing of "Result" variable and capturing of for-loop counters.

2. A FrameObject of nested function have reference to frame object of outer function. Let's call frame object of outer function "parent" FrameObject.

   Link to parent FrameObject allows to access captured variables of outer functions of any depth of nesting.

3. A FrameObject is a managed object which is reference counted. It inherits TInterfacedObject.

   Otherwise memory management will be the too complicated because:

   - 2 anonymous method can capture same data
   - anonymous method can be created in function call statement
   - anonymous method can capture variables of outer function.

4. An anonymous function is the methods of current FrameObject

   This is feature of Delphi. Such implementations allows

   - to access captured variables from anonymous method
   - all method created in same function will access same data
   - reference to method contains pointer to FrameObject. This will keep all captured variables alive because of reference counting.

   Such implementation have one gotcha. In example below someone can expect that each element of procArr is a function which have it's own variable innerVariable. This is not truth. innerVariable is same for each element of procArr. Moreover all elements of procArr are equal.

```
    for i := 1 to 5 do
      procArr[i] :=
        procedure
        var innerVariable: Integer;
        begin
        end;
```

5. There is no way to capture variable by value.

   Delhi's documentation is clear: "Note that variable capture captures variables–not values"

6. Anonymous method captures not only variables used in it's body but also all variables used in nested functions and nested anonymous methods.

   Reason: during call of nested function or creation of nested anonymous method all required variables should be alive.

7. Delphi sometimes creates FrameObject for function which have neither captured variables not anonymous methods.

   Only FrameObject of nearest outer function can be used as parent FrameObject to access captured variables of outer functions(of any depth of nesting).

   Look at example:

```
type TProc = reference to procedure;
procedure Call(p: TProc); begin p(); end;

procedure Outer;
  procedure Nested;
    procedure NestedFactory;
    begin
      Call( procedure begin Writeln(1); end );
    end;
  begin
    NestedFactory;
  end;
begin
  Nested();
end;
```

Frame object will be created only for NestedFactory;
But in this example:

```
procedure Outer2;
var i: Integer;
  procedure Nested;
    procedure NestedFactory;
    begin
      Call( procedure begin Writeln(i); end );
    end;
  begin
    NestedFactory;
  end;
```

```
begin
  i := 10;
  Nested();
end;
```

Each function will have FrameObject. Even "Nested" function which have neither captured variables not anonymous methods.

6 More details

- Q Can i create many instances of same anonymous method in cycle?

  A No. Look at code below:

  ```
  program Simple; {$APPTYPE CONSOLE}

  type TProc = reference to Procedure;
  var i: Integer;
      arr: array[1..5] of TProc;
  begin
    for i := 1 to 5 do
      arr[i] :=
        procedure begin
        end;
    for i := 1 to 4 do
      Write(arr[i] = arr[i+1], ' ');
  end.
  ```

  It will produce output

  ```
  TRUE TRUE TRUE TRUE
  ```

  You can also find this test https://gist.github.com/vkevroletin/5069653 interesting.

- Q Where are local variables of anonymous method located?

  A On the stack.

- Q What frame object contains?

  A

  - inherited from TInterfacedObject fields
  - pointer to parent FrameObject
  - captured local variables of current function
  - separate VMT for each anonymous methos(see below)
  - may be something else

- Q Does reference to function contains 2 pointers. First pointer for object and second for code ?

  A No. It consists of single pointer.

Frame object may contains many methods. Pointer to method should contain 2 pointers. But single pointer is enough for interface VMT. If interface contains only 1 function then there is no need to store pointer to method. Frame object implements separate interface for each anonymous method. Reference to anonymous method is reference to interface which contains single method.

- Q Does (procedure begin end)() works?

  A No. This compiles, but in runtime you will get runtime error 216 Access violation

7 Examples of how anonymous methods can be implemented

Below are examples of almost equivalent sources with anonymous functions and without.

```
program Simple;

{$APPTYPE CONSOLE}

type TProc = reference to procedure;

var
  p : TProc;
  i : Integer;

begin
  i := 10;
  p := procedure begin
    Writeln(i)
  end;
  p();
end.
```

Since compiler doesn't support anonymous methods we will write:

```
program Simple;

{$mode objfpc}

type
  TFrameObjectL11 = class (TInterfacedObject)
    i : Integer;
    procedure ProcL13;
  end;

procedure TFrameObjectL11.ProcL13;
begin
  Writeln(Self.i);
end;

var
  p : procedure of object;
  frameObjL11 : TFrameObjectL11;
```

```
begin
  frameObjL11 := TFrameObjectL11.Create;

  frameObjL11.i := 10;
  p := @frameObjL11.ProcL13;
  p();
end.
```

Below anonymous method captures variables from current and from outer function:

```
program Nested;

{$APPTYPE CONSOLE}

type
  TProc = reference to procedure;

function Factory: TProc;
var v1: Integer;

  function NestedFactory: TProc;
  var v2: Integer;
  begin
    v2 := 20;
    Result := procedure begin
      Writeln('v1: ', v1,
              ' v2: ', v2);
    end;
  end;

begin
  v1 := 10;
  Result := NestedFactory();
end;

var
  p: TProc;

begin
  p := Factory();
  p();
end.
```

Since compiler doesn't support anonymous methods we will write:

```
program Nested;

{$mode objfpc}

type
```

```
  TProc = procedure of object;
  TFrameObjectL8 = class (TInterfacedObject)
    v1: Integer;
  end;
  TFrameObjectL11 = class (TInterfacedObject)
    v2: Integer;
    parent: TFrameObjectL8;
    procedure ProcL15;
  end;

procedure TFrameObjectL11.ProcL15;
begin
  Writeln('v1: ', Self.parent.v1,
          ' v2: ', Self.v2);
end;

function Factory: TProc;
var frameObjL8: TFrameObjectL8;

  function NestedFactory: TProc;
  var frameObjL11: TFrameObjectL11;
  begin
    frameObjL11 := TFrameObjectL11.Create;
    frameObjL11.parent := frameObjL8;
    frameObjL11.v2 := 20;
    Result := @frameObjL11.ProcL15;
  end;

begin
  frameObjL8 := TFrameObjectL8.Create;
  frameObjL8.v1 := 10;
  Result := NestedFactory();
end;

var
  p: TProc;

begin
  p := Factory();
  p();
end.
```

8 How tests was done

Size of heap was examined using GetHeapStatus.TotalAllocated function. Fact of creation of FrameObjects is confirmed by exploring assembler code.

I don't know how to get assembler code from Delphi, so assembler listings are in screenshots :) (but with few exaplanations).

Test was done on WinXP 32-bit with Delphi XE3.

Here is creation of FrameObject in procedure initialization
https://docs.google.com/file/d/0B36IYx_6MNY6S1Y0alVkRjZ6d1U/edit?usp=sharing
Here link of parent FrameObject assigned to current FrameObject
https://docs.google.com/file/d/0B36IYx_6MNY6NXNoZ3cxTzQzSEU/edit?usp=sharing
Here is example of creation of frame object for function which have neither captured variables not anonymous methods
https://docs.google.com/file/d/0B36IYx_6MNY6N0lmLUlKN1JtVGM/edit?usp=sharing
Here is source code which was used in examples above https://gist.github.com/vkevroletin/5070644