D4A Magazine

مجلة منتدى دلفي للعرب

Issue 05

العدد رقم 05

البرمجة بالمنحى للكائن OOP – خطوة خطوة

المؤشرات في دلفي – نظرة أعمق

قواعد البيانات، تعمق في الـADO

مقارنة بين مصممات التقارير

الزرّ و الحمار

تنبيه: هذه المقالة تجمع بين الجد و الهزل،
و يجب أن تقرأ وفق ذلك.

64bit

التعامل مع نظام 64 بت

فهـــرس العدد

افتتاحيـــــــــــة

Restriction (prohibited)

نجد عبارة "تقييد" عموما في اتفاقية استغلال البرامج التي تضعها شركة صاحب البرنامج من اجل حماية حقوقه من الاستغلال غير القانوني دون إعطاء مقابل لهذا الاستغلال، طبعا تعتبر حقوق فكرية مسجلة وعملية الموافقة تكون خلال تثبيت البرنامج على الجهاز و اختيار موافق على ما تنص عليه الاتفاقية.

نسبة المثبتين الذين يقومون بقراءة الاتفاقية لا يتعدى 15% و هذا راجع لعدم اخذ ما تحتويه بجدية.

اقتباس من اتفاقية استغلال دلفي XE:

Abstract: End-User License Agreement (EULA) for RAD Studio XE, Delphi XE, C++Builder XE, Delphi Prism XE, and RadPHP XE

2.1 LICENSE GRANT. Licensor grants to Licensee a non-exclusive, nontransferable, perpetual license (the "License") to install this Product within the country (or in the case of a country within the European Union within the European Union) specified by Licensee's ship to address provided by Licensee in the ordering documentation for the Product at the time of purchase ("Licensed Country") and solely for the development of software programs and/or management of its internal systems and data in the following manner:

(a) If Licensee has purchased a Network Named User or Named User License, Licensee may designate one person in Licensee's organization ("Named User") the right to install the Product on one or more computers and use the Product within the Licensed Country, provided that only the Named User uses the Product.

(b) If Licensee has purchased a Concurrent Users License, Licensee may install the Product on a network within the Licensed Country to be used concurrently on different computers by up to the authorized number of users for which Licensee has purchased a license provided that the Product is accessed and used only in the Territory. "Territory" means the geographical area in which the Product may be accessed and used. The use in the Territory shall be subject to the export restrictions set forth below. Territory may be any one, and only one, of the following three geographic areas: Americas Territory, EMEA Territory or Territory AsiaPac each as defined below.

The geographic Territories are:

'Americas Territory' including and limited to those geographical areas found within the boundaries of North and South America (but excluding Cuba). 'Europe, Middle East and Africa Territory' or 'EMEA Territory' including and limited to those geographical areas found within the boundaries of Europe, Middle East and Africa, including countries in the former Soviet Union (but excluding Syria, Iran and Sudan); 'Asia Pacific Territory' or 'AsiaPac Territory' including and limited to those geographical areas found within the boundaries of Asia and Australia/Pacific (but excluding North Korea).

Except where prohibited by applicable law, transfer of the Product into a country (or in the case of the European Union, outside the EU) not identified on the ordering documentation at the time of purchase is prohibited and will void the license. Temporary usage of a Product outside the Licensed Country or Territory not to exceed 30 days while a user is traveling, is permitted.

يعلم أقلية من أعضاء منتدى دلفي للعرب أن شركة Embarcadero لا تسمح ببيع منتجاتها لبعض البلدان العربية لأسباب سياسية طبعا و الكل يعلم ذلك، إن كنت من احد البلدان المعنية بالتقييد فلن تستطيع شراء دلفي من احد الوكلاء المعتمدين في العالم.

الكاتب: إدارة المنتدى

المؤشرات في دلفي – نظرة أعمق

POINTER

المؤشرات وكيفية التعامل معها أكثر ما يحير المبتدئين في البرمجة بدلفي، ذلك لأن المبتدأ يمكن أن يحزر معنى begin أو while...do وغيرها من الكلمات المحجوزة ... أما هذه الرموز @ ^ فلن يجد لها حساب!

والغريب أنه إن سأل فلا يكاد يجد من يجيب عن سؤاله، ذلك لأن الذي سيجيب غالبا ما يضطر إلى شرح مسائل متعلقة بالعتاد وتقسيم الذاكرة الحية ورسوم بيانية قد يعجز عنها...

لن نتطرق إلى كل ذلك بالتفصيل، لكن سنحاول التركيز على أهم النقاط التي تجعل منك –إن كنت مبتدئ- تنتقل إلى مرحلة أخرى وعالم آخر في البرمجة بدلفي!

تعلم المؤشرات ليس ضرورة، خاصة إن كنت تبرمج فقط تطبيقات قواعد بيانات، سهولة البرمجة بدلفي جعلتنا نتنازل في حالات كثيرة عن التعامل المباشر مع المؤشرات.

غير أن فهمك للمؤشرات سيجعلك تتحكم أكثر فيما يقوم به مترجم دلفي وبالتالي يزيد من قدرتك على اكتشاف الأخطاء أثناء التنقيح.

ملفات المساعدة في دلفي تشرح بشكل مفصل كل ما يتعلق بالمؤشرات، فلا تتردد في مراجعتها.

لندخل الموضوع...

لنمثل الذاكرة على شكل مجموعة كبيرة من الأسطر، حيث ينقسم كل سطر إلى مجموعة من الخلايا نسميها bytes، في دلفي 32 bit يصل عدد الخلايا التي يمكن أن يحجزها برنامج إلى 2^{31} خلية وهو ما يعادل 2Gb. لا تحتوي هذه الخلايا إلا على أعداد كما قد تكون خالية لا تحتوي أي قيمة.

ماذا تمثل هذه الأعداد؟

هذه الأعداد ليس لها معنى محدد ! إذ يمكن استخدامها لأي معنى مناسب، مثلا: القيمة 97 التي تخزن في خلية من الذاكرة يمكن اعتبارها عدد نوعه Byte في دلفي، كما يمكن أن استخدامها لعرض الحرف a الذي يتمثل بالقيمة 97.

عند الإعلان عن أي متغير (Variable) في برنامجك؛ يتم حجز (تخصيص) حيز بمقدار خلية (أو أكثر حسب حجم المتغير) من الذاكرة حيث يمكنك القراءة منها والكتابة فيها، يتم ذلك في المساحة المحجوزة لبرنامجك من الذاكرة، ومن خصائص هذا المتغير أن له اسما ونوعا وقيمة و**عنوان**.

```
Program Test;
Var Var1, Var2: Byte;
Var3: Integer;
Begin

end.
```

لتوضيح الفكرة سنحاول أن نمثل (بشكل خاطئ!) كيف يتم حجز حيز (أو أكثر) لكل متغير:

الخلية	...	الخلية6	الخلية5	الخلية4	الخلية3	الخلية2	الخلية1	...
المتغير		Var3				Var2	Var1	
القيمة		150				45	12	

تمثل كل خانة 1 byte (بالفرنسي: 1 Octets)، طبعا المتغيرات لا تصطف بهذا الشكل في الذاكرة، وليس شرطا أن تكون متتابعة بهذا الترتيب.

لاحظ أن المتغير Var3 يحجز (4 bytes) أربع أضعاف ما يحجزه (1 byte) المتغير Var1 ذلك لأن حجم نوع بيانات المتغير Var3 هو Integer ويعادل أربع أضعاف حجم النمط Byte، لاحظ:

```
ShowMessage(IntToStr(SizeOf(Integer)));
ShowMessage(IntToStr(SizeOf(Byte)));
```

لكن البرنامج لا يعرف معنى Var1 و Var3 وإنما يعرف كل حيز من خلال **عنوان**.

إذن، كيف يحدد البرنامج موضع المتغيرات في الذاكرة؟

يحدد البرنامج موضع أي كائن (متغير، ثابت، جدول، سجلّ ، إجراء ...) من خلال **عنوان** (**Address**) أول خلية من الحيز الذي يشغله الكائن من الذاكرة.

والعنوان هو قيمة عددية يعبر عنها –اصطلاحا- بأعداد ست-عشرية، مثال: E76013

عند الإعلان عن متغير رقمي في دلفي يتم تهيئته بقيمة 0 إذا متغير عام (Global Variable)، أما إذا كان متغير محلي (Local Variable) فلن يتم تهيئتها وستأخذ قيمة عشوائية تمثل محتوى الحيز المشغول من الذاكرة لأجل المتغير.

ماذا نعني بالمؤشرات؟

المؤشر (Pointer) هو نوع من أنواع المتغيرات الرقمية حجمه 4 byte (تماما مثل النمط Cardinal)، غير أنه لا يستخدم لتخزين البيانات وإنما لتخزين أرقام خانات من الذاكرة نسميها **عناوين**.

حسنا، كيف نحصل على عنوان المتغير Var2 مثلا؟

نحصل في دلفي على عنوان أي كائن باستخدام الرمز @ أو التابع المضمن **Addr**، لنشاهد عنوان Var2:

```
ShowMessage(IntToStr(Integer(@Var2)));
```

أو لعرضه بشكل أرقام ست-عشرية (Hexadecimal):

```
ShowMessage(IntToHex(Integer(Addr(Var2)), 8));
```

استخدمنا التغليف Integer() لأن التوابع IntToStr وIntToHex لا تقبل تمرير قيم مؤشرات مباشرة، وطبعا، العنوان يختلف من جهاز إلى آخر...

الآن، ماذا لو قمنا بتخزين عنوان المتغير Var2 في متغير آخر PVar2 ؟

في هذه الحالة نقول أن المتغير PVar2 **يؤشر على المتغير** Var2، ونمثل ذلك في الجدول:

العنوان	...	عنوان6	عنوان5	عنوان4	عنوان3	**عنوان2**	عنوان1	...
المتغير		PVar2				Var2	Var1	
القيمة		**عنوان2**				45	12	

انتبه: المتغير PVar2 يسمى **مؤشر** (Pointer) وهو لا يخزن قيمة المتغير Var2 وإنما عنوانه، ونقوم بذلك بشيء مثل:

```
program Test;
var Var1, Var2: byte;
    Var4: Pointer;
begin
  Var4 := @Var2;
end.
```

يحتمل Pointer أو المؤشر قيم موجبة (Unsigned) –نظريا- من 0 إلى 4294967295 بالأعداد العشرية، أو بأرقام ست-عشرية: من 0 إلى FFFFFFFF وهو نفس حجم البيانات Cardinal أو DWORD.

هل يمكننا الحصول على قيمة المتغير Var2 من خلال المؤشر PVar2 ؟

أجل، ببساطة نقرأ محتوى المتغير الذي عنوانه PVar2، وكأن المؤشر PVar2 يمثل مقبض للمتغير Var2. يمكننا في دلفي قراءة المتغير الذي عنوانه قيمة المؤشر PVar2 بالرمز **PVar2^**

```
var
  Var2: Byte;
  PVar2: Pointer;
begin
  Var2 := 2;
  PVar2 := @Var2;
  ShowMessage(IntToStr(Byte(PVar2^)));
```

كما يمكننا تغيير قيمة المتغير Var2 عبر مؤشره PVar2:

```
Byte(PVar2^) := 3;
```

المؤشرات قسمان:

Untyped Pointer: أي المؤشرات التي لا تؤشر على نمط محدد، ويمكن استخدامها للتأشير على أي كائن، وتعرف بالنوع Pointer : P.

Typed Pointer: وهي المؤشرات التي تستخدم للتأشير على نوع محدد من البيانات، وتعرف بأحد أنواع البيانات يسبقه الرمز ^، مثال: ^Byte ، ^Integer، ^Char، ^Boolean...

معظم هذه الأنواع معرفة في دلفي مسبقا بلاحقة P في المكتبة system.pas وهي على الترتيب: PByte، PInteger، PChar، PBoolean...

قمنا بتعريف مؤشر (PVar2: Pointer) ليس له نمط محدد (Untyped Pointer)، لذلك استخدمنا هنا التغليف إلى Byte، لنخبر مترجم دلفي أن نوع القيمة التي يؤشر عليها PVar هي Byte.

نفس النتيجة يمكن الحصول عليها بتغليف المؤشر (وليس القيمة) إلى PByte لذلك يجب تغيير موضع القوس:

```
  ShowMessage(IntToStr(PByte(PVar2)^));
```

لتحديد نوع القيمة التي يؤشر عليها PVar2 نقوم بتعريف المؤشر بنوع **مؤشر على Byte:**

```
var
  Var2: Byte;
  PVar2: ^Byte;
begin
  PVar2 := @Var2;
  PVar2^ := 5;
  ShowMessage(IntToStr(Var2));
```

ملاحظة: تغيير نوع البيانات الذي يؤشر عليه PVar2 لا يغير من حجم المؤشر، بعبارة أخرى وزن المؤشر PVar2 سيبقى 4 bytes.

هل يمكننا إنشاء مؤشر على المؤشر PVar2 ؟

طبعا وفق نفس المبدأ:

```
var
  Var1, Var2: Byte;
  PVar2: ^Byte;
  PPVar2: ^Integer;
begin
  Var1 := 2;
  Var2 := 3;
  PVar2 := @Var2;
  PPVar2 := @PVar2;
  Pointer(PPVar2^) := @Var1;
  ShowMessage(IntToStr(Byte(PVar2^)));
```

بقي شيء أخير، بما Pointer هو نفسه Integer، إذن يمكننا تخزين عنوان متغير في متغير آخر؟

لا، احذر Pointer ليس Integer (رغم أنهما بنفس الحجم، تذكر أيضا أن Integer مجال يشمل قيم سالبة) وهذه الكتابة خاطئة:

```
var
  Var1: Byte;
  Var2: Integer;
begin
  Var1 := 3;
  Var2 := @Var1;//هنا الخطأ
```

إذن كيف يمكننا تخزين عنوان المتغير Var1 في متغير عادي Var2 ؟

ببساطة نقوم بالتغليف (Pointer إلى Integer):

```
  Var2 := Integer(@Var1);
```

والأصح هنا أن يتم الإعلان عن المتغير Var2 بنوع Cardinal وليس Integer.

وللقراءة نخبر المترجم أن المتغير Var2 يحتوي على عنوان وليس على عدد، يمكن ذلك باستخدام تابع التحويل المضمن **Ptr** والمعرف كما يلي:

```
function Ptr(Address: Integer): Pointer;
```

مثال:

```
  ShowMessage(IntToStr(Byte( Ptr(Var2)^ )));
```

ليكن PVar مؤشر على Integer. يمكننا إنشاء نسخة جديدة من PVar (عدد Integer جديد):

```
var
  PVar: ^Integer;
begin
  New(PVar); //إنشاء نسخة جديدة
  PVar^ := 3;
  ShowMessage(IntToStr(Integer(PVar^)));
  Dispose(PVar);//تحرير المؤشر
```

ملاحظة: لا يمكننا إنشاء نسخة من مؤشر Untyped Pointer على غير نوع محدد، New وDispose تستخدم مع مؤشرات على أنواع محددة Typed Pointer: PByte أو PInteger...

يمكننا إنشاء عدة نسخ باستخدام شبه-الإجراء New (أو pseudo-function) وفي كل مرة نحصل على متغير جديد لا اسم له (Anonymous)، ولا يمكننا الوصول إليه إلا من خلال مؤشره.

يمكن استخدام GetMem و FreeMem لأداء نفس الغرض، غير أن هذين الإجراءين عامين ولا يخصان نوع محدد من المؤشرات، والأفضل استخدام New و Dispose.

وإذا قمنا بإنشاء نسخة أخرى باستخدام نفس المؤشر (أو جعلناه يؤشر على متغير آخر !) فسيضيع منا عنوان المتغير الأول، وبالتالي لن نتمكن من الوصول إليه ولا القراءة منه وسيسبب نزيف (أو تسرب) في الذاكرة (Memory Leak) ويصبح حينها مؤشر يتيم، مثال:

```
var
  X: Integer;
  PI: PInteger;
begin
New(PI);
  PI^ := 5;
  X := 1;
  PI := @X;//5 هنا تضيع القيمة
  ShowMessage(IntToStr(Integer(PI^)));
...
```

إلا إذا قمنا بحفظ عنوانه في مؤشر آخر، وفي مثل هذه الحالات يمكننا استعمال الصنف **TList** الذي يمكن استخدامه كجدول من المؤشرات:

```
var MyList: TList;

procedure TForm1.Button1Click(Sender: TObject);
var MyByte: ^Byte;
begin
  MyList := TList.Create;
  try
    New(MyByte);
    MyByte^ := 45;
    MyList.Add(MyByte);
    ShowMessage(IntToStr(Byte(MyList[0]^)));
    Dispose(MyByte);
  finally
    MyList.Free;
  end;
end;
```

لكن، إذا قمنا هنا بتحرير المؤشر MyByte (باستخدام Dispose) قبل عرض الناتج فسنضيع القيمة 45 (مع اختلاف في نسخ دلفي 2006 فما فوق التي تسير الذاكرة بطريقة مختلفة)، مع التنبيه إلى أن عدم تحرير المؤشر MyByte هنا يسبب نزيف في الذاكرة (يمكن فحص ذلك بضبط المتغير ReportMemoryLeaksOnShutdown على القيمة True في إصدارات دلفي 2006 فوق).

من استخدامات المؤشرات:

فهمك استخدام المؤشرات يجعلك تتحرر من قيود كثيرة تواجهها أثناء البرمجة بدلفي، يشعر بهذه القيود بعض من يستخدم لغات برمجة لا تدعم استخدام المؤشرات.

استخدام المؤشرات يمنحك تحكم أكثر في اللغة ويكسر حدودك البرمجية، مثال:

نعلم أن المتغيرات (Variables) يمكن أن تأخذ قيم مختلفة أثناء تشغيل البرنامج، خلافا للثوابت (Constants) التي تبقى قيمتها ثابت طيلة زمن تشغيل البرنامج...

ليس بعد الآن...

```
procedure ChangeConst(const Constant; var Value; Size: Integer);
begin
  Move((@Value)^, (@Constant)^, Size);
end;

procedure TForm1.Button1Click(Sender: TObject);
const
  ConstStr: string = 'String Value';
var
  VarStr: string;
begin
  VarStr := 'New String Value';
  ShowMessage(ConstStr);
  ChangeConst(ConstStr, VarStr, SizeOf(String));
  ShowMessage(ConstStr);
end;
```

بما أن المؤشر هو متغير رقمي، فيمكن إجراء معظم العمليات عليه، إرفاق قيمة: p1 := p2، الزيادة من قيمته: Inc(p) أو الإنقاص منها: Dec(p)، مع التنبيه على أن زيادة قيمة المؤشر p بمقدار D يعني إزاحة العنوان بمقدار: D×(حجم المتغير الذي يؤشر عليه p)، مثال:

```
var
  P: PDouble; // حجم الخانة التي يؤشر عليها 8
begin
  P := Ptr($50000); // تهيئة المؤشر بقيمة ابتدائية
  Inc(P);    // P = $50000 + 1 * SizeOf(Double) = $50008
...
  Inc(P, 6); // P = $50008 + 6 * Sizeof(Double) = $50038
```

لذلك شاع استخدام المؤشرات مع الجداول أو السلاسل النصية ضمن حلقة تكرارية، بالمرور على جميع القيم من خلال التأشير على الخانة اللاحقة مع كل زيادة في قيمة المؤشر.

التمرير باستخدام قيمة (مؤشر) مرجعية:

في المثال الآتي نود تغيير قيمة المتغير x من خلال إجراء بدائي:

```
procedure ChangeValue(i: Integer);
begin
  i := 0;
end;

procedure TForm1.Button1Click(Sender: TObject);
var
  x: Integer;
begin
  x := 5;
  ChangeValue(x);
  ShowMessage(IntToStr(x));
end;
```

طبعا لن تتغير قيمة x لأن التمرير في الإجراء ChangeValue تم باستخدام القيمة، أي أن قيمة المتغير i أخذت قيمة x وقام الإجراء بتغيير قيمة i إلى 0، بينما تبقى قيمة x على حالها.

طبعا، سنقول جميعا أن الحل سهل بإضافة التوجيه **var** (الذي يستخدم نفس المبدأ):

```
procedure ChangeValue(var i: Integer);
```

في لغات برمجة أخرى (لغة C مثلا) لا يوجد تمرير بهذه الطريقة، وبالتالي كان ينبغي تمرير المؤشر عوضا عن قيمة المتغير ثم التعديل من خلال المؤشر، وهو ما يقابل في دلفي:

```
procedure ChangeValue(i: PInteger);
begin
  i^ := 0;
end;

procedure TForm1.Button1Click(Sender: TObject);
var
  x: Integer;
begin
  x := 5;
  ChangeValue(@x);
  ShowMessage(IntToStr(x));
end;
```

لاحظ الآن بعد أن قمنا بتمرير عنوان المتغير x إلى الإجراء ChangeValue ليقوم هذا الأخير بتغيير قيمة محتوى العنوان الذي يحمله المؤشر i.

لماذا كل هذا ؟ ويمكننا حل المشكل باستخدام الموجه var بسهولة !

لأن هذا سيمنحك قدرات أخرى للتحكم أكثر في سلوك البرنامج، وسندرك أيضا لماذا عندما نواجه استخدام بعض دوال Windows API.

أيضا تعرف المؤشرات في لغة C بإضافة نفس الرمز (*) الذي يستخدم للتأشير، مثال:

```
  int *p, x = 5;
  p = &x;
  printf("Addr: 0x%p = %d\n", p, *p);
```

في دلفي، تم التفريق بين صيغة الاستخدام وبين صيغة التعريف:

```
var
  P: ^Integer;
  X: Integer;
begin
  X := 5;
  P := @X;
  ShowMessage(Format('Addr: $%p = %d', [p, p^]));
```

في دلفي غالبا ما تستخدم السجلات (Records) من خلال مؤشر على السجل وفق نفس المبدأ السابق، مثال:

```
type
  PPerson = ^TPerson;
  TPerson = record
    Name: string[80];
    Age: byte;
  end;

procedure SetAge(APerson: PPerson; AValue: Byte);
begin
  APerson^.Age := AValue;
end;

procedure TForm1.Button1Click(Sender: TObject);
var
  Person: TPerson;
begin
  Person.Name := 'Amine';
  Person.Age := 62;
  SetAge(@Person, 65);
  ShowMessage(IntToStr(Person.Age));
end;
```

ملاحظات:

-في هذا المثال الأخير قمنا بالإعلان عن مؤشر (PPerson = ^TPerson) على السجل TPerson قبل تعريف السجل نفسه! وهذا غير مشروع أصلا، لاحظ:

```
type
PPerson = ^TPerson;
TPerson = record
...
```

غير أنه ممكن في هذه الحالة، وهو من إحدى خصائص المؤشرات، لكن يُشترط فيه أن يكون ضمن نفس الحيز type، هذه الخاصية تستخدم بكثرة، خاصة عند تعريف سجل إحدى عناصره مؤشر لنفسه، مثال:

```
type
  PPerson = ^TPerson;
  TPerson = record
    Name: string[80]; Age: byte; Son: PPerson;
  end;
```

-أيضا في هذا المثال يمكننا الاستغناء عن المؤشر وتمرير متغير من السجل:

```
procedure SetAge(var APerson: TPerson; AValue: Byte);
```

غير أن الإجراء SetAge هنا سيقوم بإنشاء نسخة كاملة من السجل TPerson لتغيير قيمة وحيدة Age، وهو ما يجعل الأمر أثقل خاصة مع ضخامة حجم السجل وكثرة الإجراءات... تمرير المؤشر في هذه الحالة أنظف وأسرع.

-بخصوص ^ يمكن إهماله في حالة التعامل مع السجلات ونكتفي مباشرة بـ:

```
  APerson.Age := AValue;
```

إعدام المؤشر:

إذا جعلنا المؤشر p2 يأخذ قيمة المؤشر p1 الذي يؤشر على متغير x وقمنا بتحرير قيمة المؤشر p1 باستخدام Dispose، فسيضيع المؤشر p2 ونحصل على رسالة الخطأ: Invalid pointer operation، ذلك لأن المؤشرات لا تعترف بفرض التكامل المرجعي ☺، مثال:

```
var
  x: Integer; p1, p2: ^Integer;
begin
  x := 5;
  p1 := @x;
  p2 := p1;
  Dispose(p1);
  ShowMessage(IntToStr(p2^));//هنا الخطأ
```

بجعل المؤشر p1 يأخذ قيمة **nil** يمكننا القول أنه لا يؤشر على شيء (وسيأخذ القيمة 0 في كل دلفي إلى حد الآن) غير أنه ما يزال موجود ! وبالتالي نحصل على قيمة المتغير x في المثال السابق:

```
var
  x: Integer;
  p1, p2: PInteger;
begin
  x := 5;
  p1 := @x;
  p2 := p1;
  p1 := nil;
  ShowMessage(IntToStr(p2^));//تمت بنجاح
```

قد تكون للمؤشر قيمة (تختلف عن **nil**) لعنوان خلية محددة من الذاكرة، غير أن هذه الخلية من الذاكرة قد لا تحمل أي قيمة، وبالتالي لا يمكن الجزم بأن حتما لكل مؤشر قيمة يؤشر عليها في الذاكرة !

المؤشرات مثل بقية المتغيرات تحتاج إلى تهيئة (قيمة ابتدائية)، وإلا فستأخذ قيمة عشوائية قد تتسبب في حدوث أخطاء غير متوقعة (Access Violation) إن حاولت الكتابة من خلالها، خاصة إن كانت تؤشر خارج الحيز المحجوز للبرنامج في الذاكرة، أو قد تتسبب في تغير قيم كائنات أخرى في البرنامج.

كل الكائنات هي مؤشرات !

الصنف TObject هو كائن له مجموعة من المناهج (إجراءات وتوابع)، حيث يمكنك إنشاء نسخة منه في وقت، وتحريره من الذاكرة عند الانتهاء منه...

الكائن TObject هو في الواقع **مؤشر** إنشاء نسخة منه يعني استخدام الإجراء New على مؤشره وتحريره من الذاكرة يعني استخدام الإجراء Dispose...

أي محاولة لاستخدام نسخة من كائن TObject لم تُنشأ بعد، تعني محاولة التأشير على عدم وبالتالي: رسالة الاستثناء Access Violation...

عدم تحرير أي كائن TObject قد يعني عدم تحرير مؤشر باستخدام Dispose أي تركه معلق في الذاكرة في مكان مجهول لا يمكن الوصول إليه !

الكتابة: Object1 := Object2 (حيث كلامها كائن TObject) تعني جعل الغرض Object2 يؤشر على نفس القيمة التي يؤشر عليها الغرض Object1 ولا تعني نسخ القيم !

كل الأصناف التي تشاهدها في دلفي تنحدر من سلالة الصنف الأب TObject

بما فيها TForm، TControls، TComponent، TButton... وينطبق عليها نفس الحديث...

نفس الكلام عن الإجراءات والتوابع كلها في الواقع **مؤشرات**...

أتذكر طريقة الاستدعاء الديناميكي لمناهج مكتبات الربط الديناميكية (DLL)؟

```
Type
  TDLLProc = procedure(var Param: String); //...نمط مؤشر على إجراء
Var
  hDLL   : Integer;
  MyProc: TDLLProc; //...مؤشر على الإجراء
  S      : String;
begin
  hDLL := LoadLibrary('MyDLL.dll');
       try
         @MyProc := GetProcAddress(hDLL, 'DLLProc'); // ...حفظ قيمة مؤشر
         If @MyProc <> Nil Then
           MyProc(S);
       finally
         FreeLibrary(hDLL);
       end;
end;
```

كل ما في الأمر أن دلفي يسهل علينا البرمجة ويريحنا من تعب التعامل المباشر مع المؤشرات، في حين يمكننا –إذا استلزم الأمر- تحويل القيادة إلى طريقة أكثر يدوية...

ملاحظة:

لم نتطرق إلى الجانب العملي الذي يبين الفائدة العملية من المؤشرات، وسنترك ذلك لمواضيع لاحقة إن شاء الله، حيث سنتحدث عن بعض أنواع المؤشرات التي لها معاملة خاصة ولم نتطرق إليها في الموضوع.

تعمق في الـADO الجزء الأول

أغلب المشاكل التي يواجها المبرمجون عند برمجة التطبيقات بالـ ADO هي من سوء اختيار و استعمال الـ Cursor لأن اختيار الـ Cursor الصحيح لها تأثير مباشر على نجاح التطبيق المبرمج بالـ ADO وفهمها مهم للغاية ، نذكر بعض من المشاكل :

- التسجيلات المضافة من طرف مستخدم لا تظهر عند آخر
- بطئ فتح الاتصال
- رسالة خطأ أثناء تعديل أو حذف تسجيل معدل أو محذوف من طرف مستخدم آخر
- لا يمكن الرجوع للخلف
- للقراءة فقط
- قيمة الـ RecordCount هي -1

لذا يجب على المبرمجين أن يكونوا على دراية كافية بـالخصائص : CursorType, CursorLocation, LockType لإنشاء تطبيق أكثر كفاءة .

نبدأ بشرح بعض المصطلحات مستعملة بكثرة Recordset، Cursor، DataSet :

Recordset : ' مجموع التسجيلات '

Recordset object يحتوي نتائج الإستعلام ، النتائج تتكون من سطور (rows) تسمى تسجيلات (records) و أعمدة (columns) تسمى حقول (fields) ، كل الأعمدة تخزن في Field object في مجموعة حقول الـ Recordset .

حين يستقبل تطبيق قواعد البيانات ADO السطور من قاعدة البيانات يقوم Recordset object بتغليف البيانات و العمليات المسموحة على هذه البيانات .

Cursor :

هي هيكل البيانات الذي يخزن نتائج الإستعلامات ، في الـ ADO يمكن القول أن الكائن Recordset عبارة عن كائن COM يسهل الوصول إلى البيانات في الـ Cursor و الـ CursorType هو الذي يحدد الوظائف المتاحة للكائن Recordset .

يستعمل الـ Cursor لإحتواء مجموعة السطور المنطقية المحتفظ بها للـ Recordset

DataSet : 'مجموع البيانات'

الوحدة الأساسية للوصول إلى البيانات هي الـ DataSet وهي عائلة من الكائنات . تطبيقك يستخدم الـ DataSet للوصول لقاعدة البيانات.

كائن الـ DataSet يمثل مجموعة من السجلات من قاعدة بيانات منظمة في جدول منطقي. قد تكون هذه السجلات من جدول قاعدة بيانات واحد، أو أنها قد تمثل نتائج تنفيذ استعلام أو إجراء مخزن.

جميع الكائنات الـ DataSet التي تستخدمها في تطبيقات قواعد البيانات تنحدر من DB.TDataSet، و ترث حقول البيانات، والخصائص، والأحداث ، و المناهج من هذه الفئة ، نذكر منها :

,,, TBDEDataSet / TcustomADODataSet / TcustomSQLDataSet

TcustomADODataSet ينحدر منها :

(TADODataSet, TADOTable , TADOQuery , TADOStoredProc) كلها تشترك في الخصائص التالية :

CursorType, CursorLocation, LockType, MarshalOptions

الوصول للـ Recordset من TcustomADODataSet :

يوفرالـ TcustomADODataSet.Recordset الوصول المباشر إلى الكائنRecordset في الـ ADO.
الـRecordset هي الواجهة التي يتم من خلالها الوصول إلىRecordset لـ ADO. عند فتحADO dataset ، يتم تلقائيا تعيين قيمة Recordset إلى واجهة الذي توفر الوصول إلى السجلات. لا ينبغي أن تستخدم هذه القيمة حتى بعد إطلاق الحدث OnRecordsetCreate .
استخدام Recordset للوصول المباشر إلى إلى الكائنRecordset في الـ ADO التي يمثلها المكون dataset . مرجع الوصول المباشر هذا يسمح للتطبيق باستخدام خصائص و مناهج الكائن Recordset المصدر.

الوصول إلى كائن Recordset مفيد بشكل خاص للاستفادة من خصائص وأساليب الكائن Recordset التي لا تظهر في مكونات ADO dataset .
نادرا ما يحتاج التطبيق للوصول إلى الكائن Recordset مباشرة. نذكر حالة ممكن أن يستخدم مثل هذا الوصول : توجيه Recordset التي تنتج من تنفيذ إستعلام يرجع قيم في المكون TADOCommand. في هذه الحالة، تخصيص الـ Recordset المرجعة من تنفيذ (TADOCommand) مباشرة إلى الخاصية

Recordset. للـ ADO dataset المستعملة للإستقبال بالشكل التالي :

```
ADODataSet1.Recordset := ADOCommand1.Execute;
```

1- خاصية CursorLocation:

استخدام CursorLocation لتحديد أين يتم إنشاء الـ Recordset حين تفتح جانب -الزبون أو جانب-السيرفر.

القيم المحتملة : clUseClient و clUseServer

دلفي يغطي CursorLocation في TCursorLocation .

القيمة الإفتراضية : clUseClient - هذه القيم تعطى قبل فتح الإتصال .

يجب الأخذ بعين الإعتبار كل العوامل والموارد حين إختيار CursorLocation

clUseClient :

يتم تخزين نتائج الإستعلامات كاملة في المحرك ADO Cursor و يستعمل لإدارتها .

- السرعة و الكفاءة في التعامل مع البيانات لأنها تجلب البيانات إلى الذاكرة المحلية .
- عبارات الـ SQL تنفذ في السيرفر .
- العبارات التي تحدد مجموع التسجيلات بإستخدام Where تستقبل في local cursor (مختزلة).
- إمكانية العمل بدون إتصال (الإتصال فقط لجلب أو حفظ البيانات)
- تعرض مرونة أكبرالسماح للعمليات الغير مدعومة من طرف clUseServer مثل الترتيب و الفلترة .
- إمكانية تخزين البيانات و جلبها من الملفات (ADTG، XML).
- إرتفاع حركة مرور شبكة الاتصال (يتم تمرير البيانات من و إلى العميل).

clUseServer :

الـ Recordset تتم إدارتها بالمزود OLE DB provider و/ أو قاعدة البيانات

- إقتصاد موارد العميل .
- إستهلاك موارد السيرفر لكل زبون متصل .
- عدم الإمكانية من العمل بدون إتصال (البقاء متصل) .
- انخفاض حركة مرور شبكة الاتصال و تعتبر الأمثل في حالة ضعف الإتصال .

2 - خاصية CursorType :

لا يمكن التكلم عن Recordset بدون التكلم عن CursorType

خاصية الـ CursorType تحتوي على القيمة التي تدل على نوع الـ Cursor المستعمل لتحديد كيف تتحرك داخل السجلات و عن ظهور أو عدم ظهور التغييرات التي أدخلت في قاعدة البيانات من طرف مستعملين آخرين بعد إستقبال البيانات .

القيم المحتملة : ctstatic, ctforward-only, ctkeyset, ctdynamic، ctUnspecified

دلفي يغطي cursor types في TcursorType.

القيمة الإفتراضية هي ctKeyset - هذه القيم تعطى قبل فتح الـ dataset .

ctUnspecified لم يحدد بعد الـ CursorType.

ctDynamic

- تسمح لك برؤية التغييرات (الإضافة و تعديل و حذف) من قبل المستعملين الآخرين .
- تسمح لك بعمل تغييرات (الإضافة و تعديل و حذف).
- التنقل في كل الإتجاهات .

ctKeyset

- تسمح لك برؤية (التعديل) من قبل المستعملين الآخرين أما التسجيلات المضافة لا يمكن رؤيتها و التسجيلات المحذوفة لا يمكن الولوج إليها .
- تسمح لك بعمل تغييرات (الإضافة و تعديل و حذف).
- التنقل في كل الإتجاهات .

ctStatic يقوم بقراءة كاملة لمجموع النتائج (نسخة ثابتة من مجموع التسجيلات)

- لا تسمح لك برؤية التغييرات (الإضافة و تعديل و حذف) من قبل المستعملين الآخرين .
- التنقل في كل الإتجاهات .
- تسمح لك بعمل تغييرات (الإضافة و تعديل و حذف).
- تستخدم عادة للتقارير في حالة clUseClient : CursorLocation .

- يستعمل فقط عندما يكون clUseClient : CursorLocation

ctForward-only (للأمام فقط | عبور واحد)

- تسمح لك برؤية التغييرات (الإضافة و تعديل و حذف) من قبل المستعملين الآخرين ' إذا لم تصل بعد إلى هذه التسجيلات' .
- التنقل في إتجاه واحد " الأمام " (UniDirectional).
- سريع جدا ! بعد الإنتقال لسجل الموالي يحذف السجل السابق من الذاكرة .
- قيمة الخاصية RecordCount لإستعادة عدد التسجيلات هو -1 .

من فوائده أنه سريع و هو الأمثل لملأ ClientDataset / القوائم النصية مثل ComboBox ... .

FireHorse ' الحصان الناري' في الحقيقة هو ليس بـ CursorType سمي هكذا لسرعته .

هو عبارة عن ctForward-only مع CursorLocation = clUseServer و LockType= ItReadOnly.

إضافة إلى خصائص الـ ctForward-only :

- لا تسمح لك بعمل تغييرات (الإضافة و تعديل و حذف) "القراءة فقط ".
- سريع جدا ! و أسرع من السابق ctForward-only. لأنه لقراءة فقط .
- تستخدم عادة في التقارير في حالة CursorLocation : clUseServer

* كلها تدعم الـ bookmark إلا ctForward-only (إذا تحركت من تسجيل لا يمكن العودة إليه) .

* إذا تم طلب CursorType غير مدعوم من المزود ، فإن المزود ممكن أن يعطيك آخر مثلا إذا وضعت CursorLocation = clUseServer و CursorType = ctDynamic مع قاعدة بيانات أكسس فإنه سوف يغير CursorType إلى ctKeyset .

* في حالة CursorLocation = clUseClient الـ CursorType يحتمل قيمة واحدة ctStatic يعني أن القيم الباقية خاصة بالـ clUseServer

2 - خاصية LockType :

إذا كنت تبرمج برنامج متعدد المستخدمين فعليك الأخذ بعين الإعتبار خاصية LockType لمنع عدة مستخدمين من تعديل/حذف نفس التسجيل (إغلاق/حماية التسجيل) .

LockType تخبر المزود أي نوع من الإغلاق يجب أن يوضع على السجلات أثناء التحديث (ولإستعمالها يجب أن يكون المزود يدعمه).

القيم المحتملة : ltOptimistic، ltPessimistic، ltReadOnly، ltUnspecified، ltBatchOptimistic

دلفي يغطي LockType في TADOLockType.

القيمة الإفتراضية هي ltOptimistic - هذه القيم تعطى قبل فتح الـ dataset .

ltUnspecified لم يحدد بعد الـ LockType.

ltOptimistic

* في هذا النوع يتم غلق كل سجل على حدة فقط حين تتم عملية تحديث السجل (الفيزيائي) و قيم هذا السجل هي آخر القيم المعطاة من عند آخر مستخدم قام بالتحديث

ltPessimistic

* في هذا النوع يتم غلق كل سجل على حدة أثناء عملية التعديل حتى الحفظ (لا تدعم بعض المزودات هذا النوع).

ltReadOnly

* الإسم يعبر عن نفسه في هذه الحالة يكون في حالة القراءة فقط.

ltBatchOptimistic

* التحديث بالدفعات عوض إستخدام التحديث الفوري .

* تستعمل مع clUseClient للعمل دون إتصال (فكرتها هي أن المستخدم يقوم بالعمليات ، تخزن هذه العمليات في الذاكرة ثم تقدم على شكل دفعة (batch) حين تكون جاهزة إلى قاعدة البيانات وهذا بالتعليمة (UpdateBatch) و يمكن إلغاء ما يوجد في (batch) بواسطة CancelBatch أو CancelUpdates

ملاحظة :

يمكن أن يوجد تضارب في حالة (قيام أكثر من مستخدم بتعديل/حذف نفس التسجيل ، مثلا الأول قام بالتعديل ثم الحفظ الثاني لايمكنه الحفظ إلا بعد جلب آخر تحديث للتسجيل) .

Message

“Row cannot be located for updating.Some values may have been changed since it was last read”

"لايمكن إيجاد السطر للتعديل ، بعض القيم تغيرت قيمها بعد آخر قراءة"

نعود إلى هذه الحالة في موضوع آخر إن شاء الله ...

2 - خاصية MarshalOptions :

تستعمل مع clUseClient عند تغيير في البيانات ليتم إرسالها إلى السيرفر ،بهذه الخاصية يمكن تخصيص التسجيلات في البيانات المحلية التي يتم إرسالها إلى السيرفر .

القيم المحتملة : moMarshalAll ، moMarshalModifiedOnly

دلفي يغطي MarshalOptions في TMarshalOption.

القيمة الإفتراضية : moMarshalAll .

moMarshalAll : يتم تعبئة جميع التسجيلات في البيانات المحلية و إرسالها إلى السيرفر.
moMarshalModifiedOnly : يتم تعبئة فقط التسجيلات التي تم تغييرها في البيانات المحلية و إرسالها إلى السيرفر .

الأكسس : في الأكسس عليك العمل بما يأتي إفتراضيا أو إتبع الجدول التالي :

LockType	CursorType	CursorLocation
ltReadOnly	ctForwardOnly ctKeyset ctStatic	clUseServer
ltReadOnly ltPessimistic ltOptimistic ltBatchOptimistic	ctKeyset	
ltReadOnly ltOptimistic ltBatchOptimistic	ctStatic	clUseClient

1 - مكون ADOConnection :

-<ConnectionString> : فيها نكتب معلومات الإتصال من مزود و ملف القاعدة و كلمة المرور إن وجدت و ..

- Connected : للإتصال و قطع

- LoginPrompt : خاصة بنافذة طلب اسم المستخدم و كلمة المرور إذا كنت لا تريدها أن تظهر إجعل القيمة False

- KeepConnection : إبقاء الإتصال أو قطعه في حالة عدم وجود Dataset في وضع Active .

2 - مكون ADOTable :

-<ConnectionString>: إذا أردت إنشاء إتصال آخر دون إستعمال ADOConnection.

- AutoCalcFields : لتحديد الحقول إذا كانت الحقول الحسابية تحسب أوتوماتيكيا أم لا .

- Filter : يستعمل للفلترة (باستعمال حقل أو أكثر تقابلها القيم).

- Filtered : لتفعيل و إلغاء تفعيل الفلترة .

-MaxRecords : لتحديد عدد التسجيلات المراد إظهارها إفتراضيا 0 تعني غير محدود .

- CommandTimeout : لتحديد وقت لتنفيذ أمر ما (sql) بعد إنتهاء هذا الوقت يتم إلغاء الأمر المراد تنفيذه،وهو محدد بالثانية مثلا عند محاولة تنفيذ أمر و إنقطع الإتصال بالسيرفر سوف يلغى الأمر بعد إنتهاء الوقت المعطى .

3 - مكون ADOQuery :

- SQL : لكتابة جمل الـ SQL .

- AutoCalcFields : لتحديد الحقول إذا كانت الحقول الحسابية تحسب أوتوماتيكيا أم لا .

- Prepared: لتحديد إذا تهيئة جمل الـ SQL من عدمها .

- CacheSize : لتحديد حجم الذاكرة الوسيطة للـ dataset للتحكم في عدد الأسطر المحتفظ بها في الذاكرة من طرف المزود ، القيمة الإفتراضية هي 1 و هي أقل قيمة مسموح بها .

4 - مكون ADOStoredProc :

- ProcedureName: لتحديد الإجراء المخزن في قاعدة البيانات .

5 - مكون ADODataSet /ADOCommand :

- CommandType : لتحديد نوع الأمر الذي سينفذ (cmdTable، cmdTableDirect، cmdStoredProc، cmdFile، cmdText، cmdUnknown)

cmdFile : لفتح الملفات المحفوظة على شكل xml / adtg بواسطة الـ DataSet) تختار إسم الملف في الخاصية CommandText أو بالضغط باليمين على ADODataSet و إختر Load From File)

cmdStoredProc للتعامل مع الإجراءات المخزنة (تختار إسم الإجراء المخزن في الخاصيةCommandText)

cmdTable، cmdTableDirect للتعامل المباشر مع الجداول (تختار إسم الجدول في الخاصية CommandText)

cmdText، cmdUnknown للتعامل بجمل الـ SQL (تكتب جمل الجدول الـ SQL في الخاصية CommandText)

لكن بتحديد نوع الـ CommandType و عدم تركه cmdUnknown النتيجة تكون أداء أفضل .أيبل

ملاحظة :

cmdOpenFile, cmdTableDirect, cmdTable لا يجب أن تستخدم بـ ADOCommand .

مقارنة بين مصممات التقارير – الجزء الأول

بعض المبرمجين يفضلون توليد التقارير يدويا (بالكود) لكن الأغلبية يفضلون إستخدام مصممات التقارير لأنها تقدم واجهات لتصميم التقارير بسهولة وسرعة

أعتقد أن كل واحد من طرح التساؤل :

ما هو أحسن مصمم تقارير ؟

كيف أختار المناسب لي ؟

نقدم لكم مقارنة في بعض الوظائف الأساسية بين أشهر و أفضل مصممات التقارير في دلفي :

Rave reports

مميزاته

إصدار مجاني (Rave BE Bundled Edition) تنصب أوتوماتيكيامع الدلفي إبتداءا من الإصدار 7 إلى آخر إصدار XE .

متكامل مع بيئة التطوير دلفي.

إمكانية جمع عدة تقارير في ملف واحد.

إمكانية حمل ملف التقارير في الملف التنفيذي.

إحتوائه على مساعد في الرسم Wizard فقط للتقارير البسيطة .

محرر السكربت (للتعامل مع الأحدا ث).

التصدير إلى : pdf , rtf ,html , txt .

سلبياته

عدم إمكانية حمل مصمم التقارير في الملف التنفيذي إلا عند شراء الإصدار Rave Reports Architect بسعر 400 دولار (في الـ Rave هو عبارة عن dll).

الدعم الفني منعدم .

الموقع الرسمي ميت.

لا إصدار للدوت نت .

Report Builder

مميزاته

يدعم إصدارات الدلفي من 4 إلى دلفي XE.

متكامل مع بيئة التطوير دلفي.

السورس مرفق مع جميع الإصدارات .

Runtime Pascal Environment (RAP) أوبجكت باسكال(محرر السكربت) مع إستعمال الأحداث المدمجة لإنشاء التقارير المعقدة .

توثيق جيد.

إمكانية حمل مصمم التقارير (End-user layout Editor) في الملف التنفيذي .

سلبياته

RAP متوفر فقط في الإصدارين Server و Enterprise .

حمل مصمم التقارير (End-user layout Editor) يبدأ من إصدار Professional .

أسعار مرتفعة إبتداءا من Standard بـ 349 دولار إلى Server بـ 1099 دولار .

لا إصدار للدوت نت .

Quick Report

مميزاته

يدعم إصدارات الدلفي من 5 إلى دلفي XE .

متكامل مع بيئة التطوير دلفي .

سعر الترقية 25 % من سعر الرخصة .

يوجد end-user report designer خارجي QRDesign مجاني لمن يملك الإصدار QuickReport Pro .

يمكن تضيمنه داخل الملف التنفيذي .

تصدير التقارير على شكل : HTML, PDF, XML, CSV, XL, WMF,ASCII .

سلبياته

السعر حوالي 240 أورو للإصدار QuickReport Pro .

لا إصدار للدوت نت .

Fast Report

مميزاته

يدعم من الدلفي 4 إلى XE .

يوجد إصدار للدوت نت (سهولة تحويل التقارير من vcl إلى .net من *.fr3 إلى *.frx) .

إمكانية حمل مصمم التقارير في الملف التنفيذي يبدأ من إصدار standard دون الدفع مصاريف إضافية .

web reports في الإصدار (Enterprise) .

الأسعار تبدأ من 79 دولار (Basic) إلى 349 دولار (Enterprise) .

توفر أداة للإستيراد وتحويل تقارير Rave reports و Quick Report .

محرر السكربت قوي (PascalScript,C++Script,BasicScript,Jscript) مع الـ Debuger .

مصمم صفحات الحوار Dialog-Page .

توثيق جيد.

دعم لليونيكود (كان قبل دعم الدلفي له) .

يأتي معه 'باني الإستعلامات' Fast Query Builder .

التصدير إلى : pdf , xsl, rtf , html ,bmp , tiff , jpeg , gif, csv , txt , mail , odt , ods .

سلبياته

السورس متوفر مجانا إبتداءا من الإصدار (Professional) .

الجديد في الإصدار القادم FastReport VCL 5

- تحسينات في المحرك .
- كائنات جديدة نذكر منها :* الباركود DataMatrix و PDF417 و * Zip Code ..
- + التصدير إلى : DOCX / XLSX / PPTX / XLS BIFF .
- الواجهة بالـ Ribbon . ...

DevExpress ExpressPrinting System

مميزاته

ExpressPrinting System هو نظام متقدم لتمثيل البيانات و نظام طباعة صمم خصيصا لتقديم واجهة المستخدم إلى الصفحة المطبوعة

عبر تكنولوجيا Report Link 'وصل التقرير ' يسمح ExpressPrinting System بإخراج محتوى المتحكمات vcl مثل ExpressQuantumGrid و ExpressVerticalGrid و العديد من المكونات نذكر منها :

TPicture/TImage وTDBImage/ TMemo
وTDBMemo/TCheckListBox/TListBox وTDBListBox/TListBox
وTDBListBox/TRichEditوTDBRichEdit

TChart/ TTreeView/ TListView وTStringGrid/TDBChart...

يوفر لك القدرة على تقديم تقارير غير محدودة بسرعة لبرامجك بدون تصميم تقرير واحد !

المعاينة بثلاث طرق حسب الإختيار Standard و Advanced و Ribbon .

يدعم من الدلفي 7 إلى XE .

توثيق جيد .

سلبياته

زيادة حوالي 5 MB في حجم الملف التنفيذي .

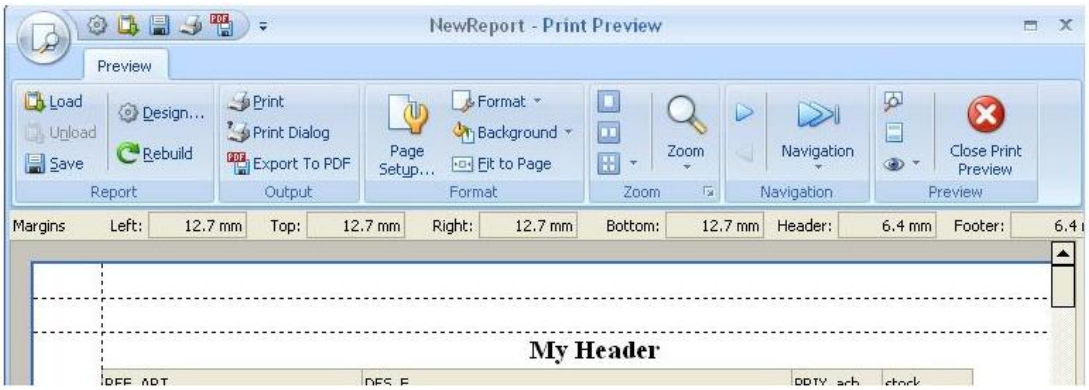
التصدير إلى Pdf فقط .

لا وجود للـ DBGrid ضمن قائمة الـ Report Link مما يجعلك تعمل بالـ TStringGrid مثلا أو شراء ExpressQuantumGrid .

السعر 199 دولار إذا قمت بشرائه على حدة .


Preview
File View Format Go
100%
Margins Left: 12.7 mm Top: 12.7 mm Right: 12.7 mm Bottom: 12.7 mm Header: 6.4 mm Footer: 6.4
My Header
REF_ART DES_F PRIX_ach stock
310100 Alphamalt A 5070 1785.89 -109.502


Print Preview
File View Insert Format Go Tools
100%
Margins Left: 12.7 mm Top: 12.7 mm Right: 12.7 mm Bottom: 12.7 mm Header: 6.4 mm Footer: 6.4
My Header
REF_ART DES_F PRIX_ach stock
310100 Alphamalt A 5070 1785.89 -109.502


NewReport - Print Preview
Preview
Load
Unload
Save
Design...
Rebuild
Report
Print
Print Dialog
Export To PDF
Output
Page Setup...
Format
Background
Fit to Page
Format
Zoom
Zoom
Navigation
Navigation
Close Print Preview
Preview
Margins Left: 12.7 mm Top: 12.7 mm Right: 12.7 mm Bottom: 12.7 mm Header: 6.4 mm Footer: 6.4
My Header

Wow64DisableWow64FsRedirection

تمهيد:

بما أن الشركة الوصية الحالية Embarcadero Technologies أظهرت نية دعم نظام 64 بت و برمجة X64 Compiler للإصدارات المستقبلية لدلفي فمن الحكمة أن نبدأ المبادرة و الاحتكاك بالنظام لمحاولة فهم آلياته و طرق عمله.

لذا فضلت أن اطرح بعض الأمثلة على شكل سلسلة مقالات تطبيقية يتم نشرها حصرا باللغة العربية في إصدارات مجلة منتدى دلفي للعرب و باللغة الانجليزية في مدونة Slug Analysis Lab.

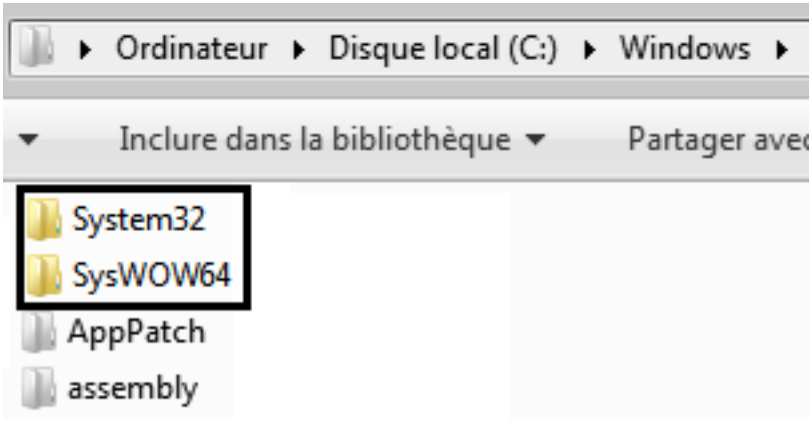
موضوع المقالة:

يتمتع نظام 64 بت بمرونة كبيرة في التعامل م تطبيقات 64 بت و 32 بت، حيث عمدت شركة مايكروسوفت إلى تقسيم سجل النظام و مجلداته إلى قسمين، القسم الأول و هو الأساسي يخص تطبيقاته الافتراضية بما انه نظام 64 بت و القسم الثاني كمرحلة انتقالية يحتوي على تطبيقات 32 بت.

و يتم معالجة الرسائل و تنفيذ الأوامر بعد الفحص و التعرف على نوعية التطبيق الذي يتم تشغيله، مثلا نبرمج تطبيقين متماثلين في الأوامر، الأول 32 بت و الثاني 64 بت و نقوم بتشغيلهما على نظام 64 بت، بعد استدعاء مدير المهام سوف نلاحظ أن النظام عمل تصنيف لتطبيقين، الأول أضاف إليه 32 و الثاني تركه على حاله.

سوف نجد نفس التصنيف في حالة تطبيق 32 بت أراد أن يتعامل مع مجلد النظام %SystemRoot%\System32، النظام يقوم بعمل Redirection إعادة توجيه بصفة افتراضية للمسار إلى مسار تطبيقات 32 بت %SystemRoot%\SysWOW64 حتى و أن كتبنا المسار بدون متغيرات النظام بصفة صريحة مثل : C:\Windows\System32 سوف يتم توجيهنا إلى هذا المسار المذكور سابقا :C:\Windows\ SysWOW64

و السبب هو التنظيم لتفادي الأخطاء المحتملة لو كان غير ذلك، طبعا شركة مايكروسوفت أعطت خيار تعطيل و تشغيل إعادة التوجيه للمبرمج حسب ما يريد انجازه من مهمة تحتاج إلى التعامل مع هذا الخيار.



للاستزادة اقتباس من موقع MSDN

Wow64DisableWow64FsRedirection **Function**

Disables file system redirection for the calling thread. File system redirection is enabled by default.

```
Syntax
BOOL WINAPI Wow64DisableWow64FsRedirection(
  __out  PVOID *OldValue
);
```

```
Parameters
OldValue [out]
```

The WOW64 file system redirection value. The system uses this parameter to store information necessary to revert (re-enable) file system redirection.

Note This value is for system use only. To avoid unpredictable behavior, do not modify this value in any way.

Return Value

If the function succeeds, the return value is a nonzero value.

If the function fails, the return value is zero. To get extended error information, call **GetLastError**.

Remarks

This function is useful for 32-bit applications that want to gain access to the native system32 directory. By default, WOW64 file system redirection is enabled.

The **Wow64DisableWow64FsRedirection**/**Wow64RevertWow64FsRedirection** function pairing is a replacement for the functionality of the **Wow64EnableWow64FsRedirection** function.

To restore file system redirection, call the **Wow64RevertWow64FsRedirection** function. Every successful call to the **Wow64DisableWow64FsRedirection** function must have a matching call to the **Wow64RevertWow64FsRedirection** function. This will ensure redirection is re-enabled and frees associated system resources.

Note The **Wow64DisableWow64FsRedirection** function affects all file operations performed by the current thread, which can have unintended consequences if file system redirection is disabled for any length of time. For example, DLL loading depends on file system redirection, so disabling file system redirection will cause DLL loading to fail. Also, many feature implementations use delayed loading and will fail while redirection is disabled. The failure state of the initial delay-load operation is persisted, so any subsequent use of the delay-load function will fail even after file system redirection is re-enabled. To avoid these problems, disable file system redirection immediately before calls to specific file I/O functions (such as **CreateFile**) that must not be redirected, and re-enable file system redirection immediately afterward using **Wow64RevertWow64FsRedirection**.

Disabling file system redirection affects only operations made by the current thread. Some functions, such as **CreateProcessAsUser**, do their work on another thread, which is not affected by the state of file system redirection in the calling thread.

Examples

The following example uses **Wow64DisableWow64FsRedirection** to disable file system redirection so that a 32-bit application that is running under WOW64 can open the 64-bit version of Notepad.exe in %SystemRoot%\System32 instead of being redirected to the 32-bit version in %SystemRoot%\SysWOW64.

```
#define _WIN32_WINNT 0x0501
#include <Windows.h>

void main()
{
    HANDLE hFile = INVALID_HANDLE_VALUE;
    PVOID OldValue = NULL;

    //  Disable redirection immediately prior to the native API
    //  function call.
    if( Wow64DisableWow64FsRedirection(&OldValue) )
    {
        //  Any function calls in this block of code should be as
concise
        //  and as simple as possible to avoid unintended results.
        hFile = CreateFile(TEXT("C:\\Windows\\System32\\Notepad.exe"),
            GENERIC_READ,
            FILE_SHARE_READ,
            NULL,
            OPEN_EXISTING,
            FILE_ATTRIBUTE_NORMAL,
            NULL);

        //  Immediately re-enable redirection. Note that any resources
        //  associated with OldValue are cleaned up by this call.
        if ( FALSE == Wow64RevertWow64FsRedirection(OldValue) )
        {
            //  Failure to re-enable redirection should be considered
            //  a criticial failure and execution aborted.
            return;
        }
    }

    //  The handle, if valid, now can be used as usual, and without
    //  leaving redirection disabled.
    if( INVALID_HANDLE_VALUE != hFile )
    {
        // Use the file handle
    }
}
```

مثال تطبيقي:

```
procedure TWinMain.CopyBtnClick(Sender: TObject);
var
  Wow64DisableWow64FsRedirection: function(var OldValue: Pointer):
BOOL; stdcall;
  Wow64RevertWow64FsRedirection: function(OldValue: Pointer): BOOL;
stdcall;
  OldValue: Pointer;
begin
  try
    OldValue := nil;
    CopyFile(PChar('C:\Windows\System32\Notepad.exe'),
PChar('C:\Test\Notepad32.exe'), TRUE);

    Wow64DisableWow64FsRedirection :=
GetProcAddress(GetModuleHandle(kernel32),
'Wow64DisableWow64FsRedirection');
    Wow64RevertWow64FsRedirection :=
GetProcAddress(GetModuleHandle(kernel32),
'Wow64RevertWow64FsRedirection');

    if Wow64DisableWow64FsRedirection(OldValue) then
    begin
      CopyFile(PChar('C:\Windows\System32\Notepad.exe'),
PChar('C:\Test\Notepad64.exe'), TRUE);
      MessageBox(Handle, PChar('Wow64 file system redirection
[Disabled sucessfully]'), PChar('INFORMATION'), MB_ICONINFORMATION);
    end
    else
    begin
      MessageBox(Handle, PChar('Unable to disable Wow64 file system
redirection'), PChar('ERROR'), MB_ICONERROR);
      Exit;
    end;

    if Wow64RevertWow64FsRedirection(OldValue) then
      MessageBox(Handle, PChar('Wow64 file system redirection [Enabled
sucessfully]'), PChar('INFORMATION'), MB_ICONINFORMATION)
    else
      MessageBox(Handle, PChar('Unable to Enable Wow64 file system
redirection'), PChar('ERROR'), MB_ICONERROR);
  except
   (* nothing *)

  end;
end;
```

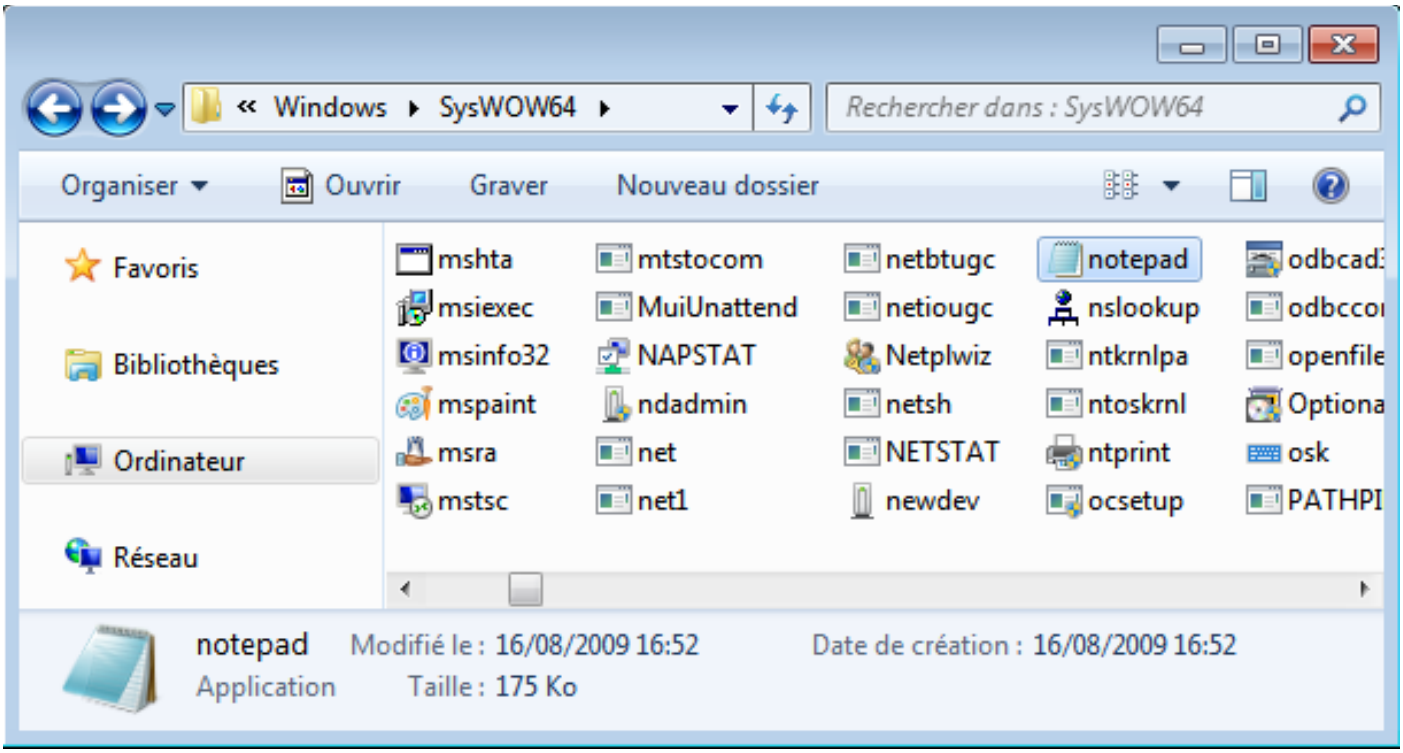
التعليق على المثال:

الجزء الملون باللون الأخضر يخص تعريف دالتين تصدرهما مكتبة Kernel32.dll الموجودتين فقط على نظام 64 بت و طريقة الحصول على عنوانهما من المكتبة.

الجزء الملون باللون البني يخص استدعاء دالة التعطيل و إعادة التفعيل.

الجزء الملون باللون الأزرق يخص عملية النسخ قبل و بعد معالجة خاصية إعادة التوجيه.

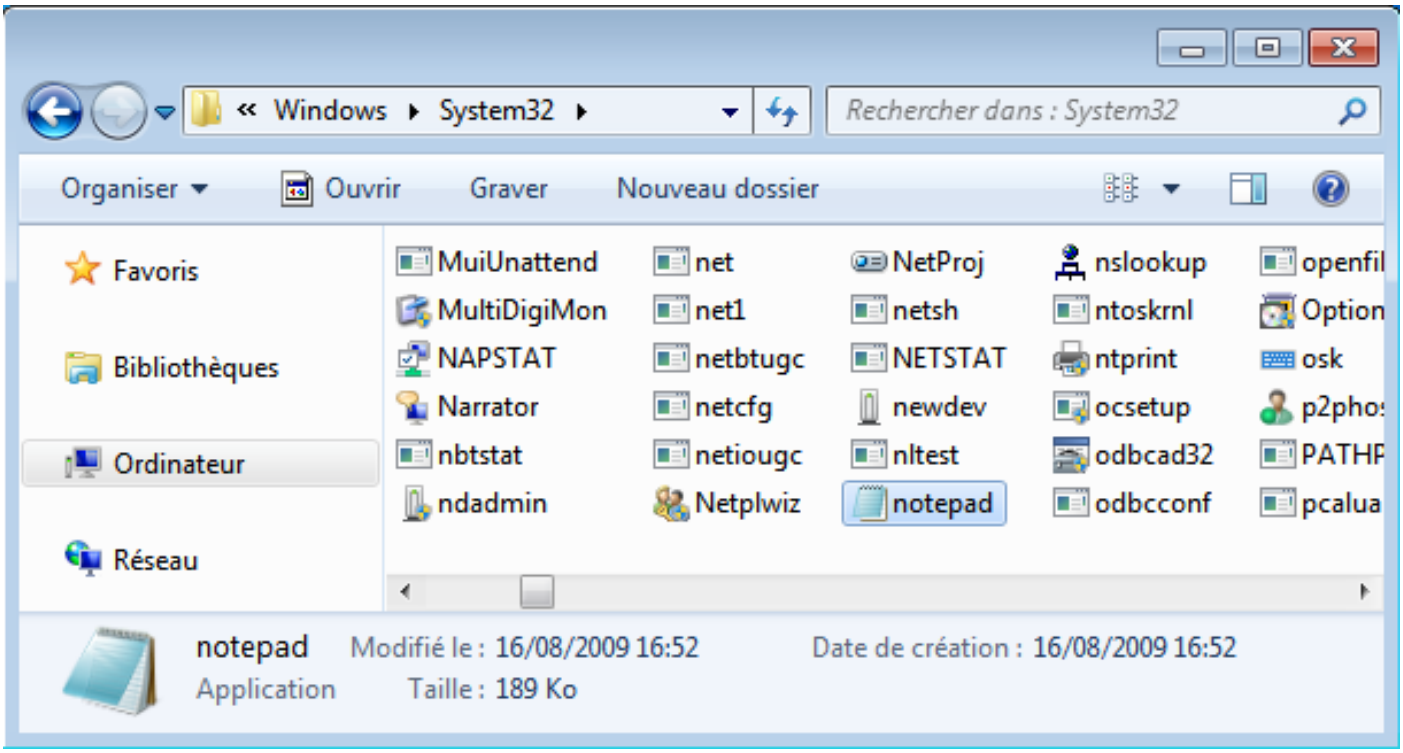
التطبيق يقوم بمحاولة نسخ ملف notepad.exe من مجلد النظام C:\Windows\System32 إلى مجلد C:\Test بدون تعطيل إعادة التوجيه، فيقوم النظام بتوجيه المسار إلى المجلد الخاص بتطبيقات 32 بت C:\Windows\ SysWOW64



فنحصل على ملف 32 بت، ثم يكمل التطبيق العملية بتعطيل إعادة التوجيه.

نلاحظ أن دالة تعطيل إعادة التوجيه تطلب متغير- مخرج - للكي يتم فيه حفظ حالة التوجيه قبل تعطيلها، و يتم تمرير هذه الأخيرة - القيمة المرجعة - فيما بعد لدالة تفعيل إعادة التوجيه.

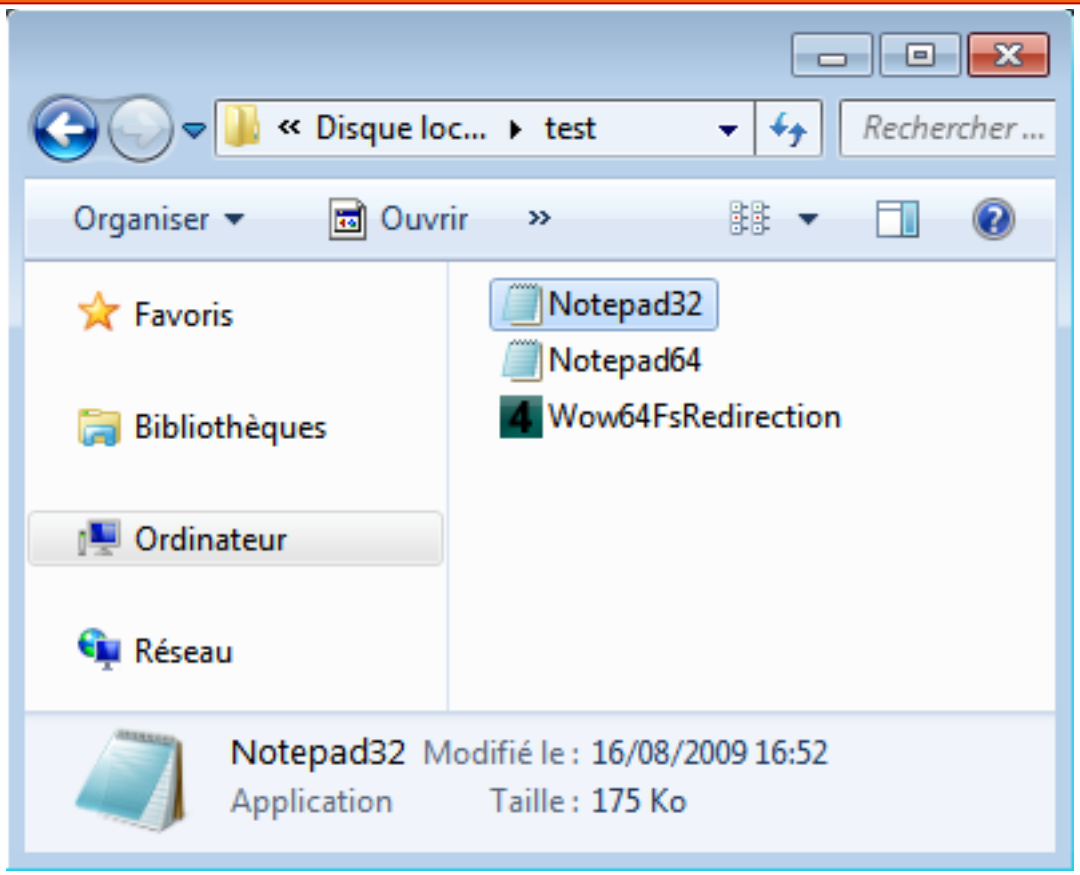
يتم نسخ الملف من جديد فنحصل على ملف ثاني و لكن هذه المرة ملف 64 بت من المسار الافتراضي للنظام C:\Windows\System32 .

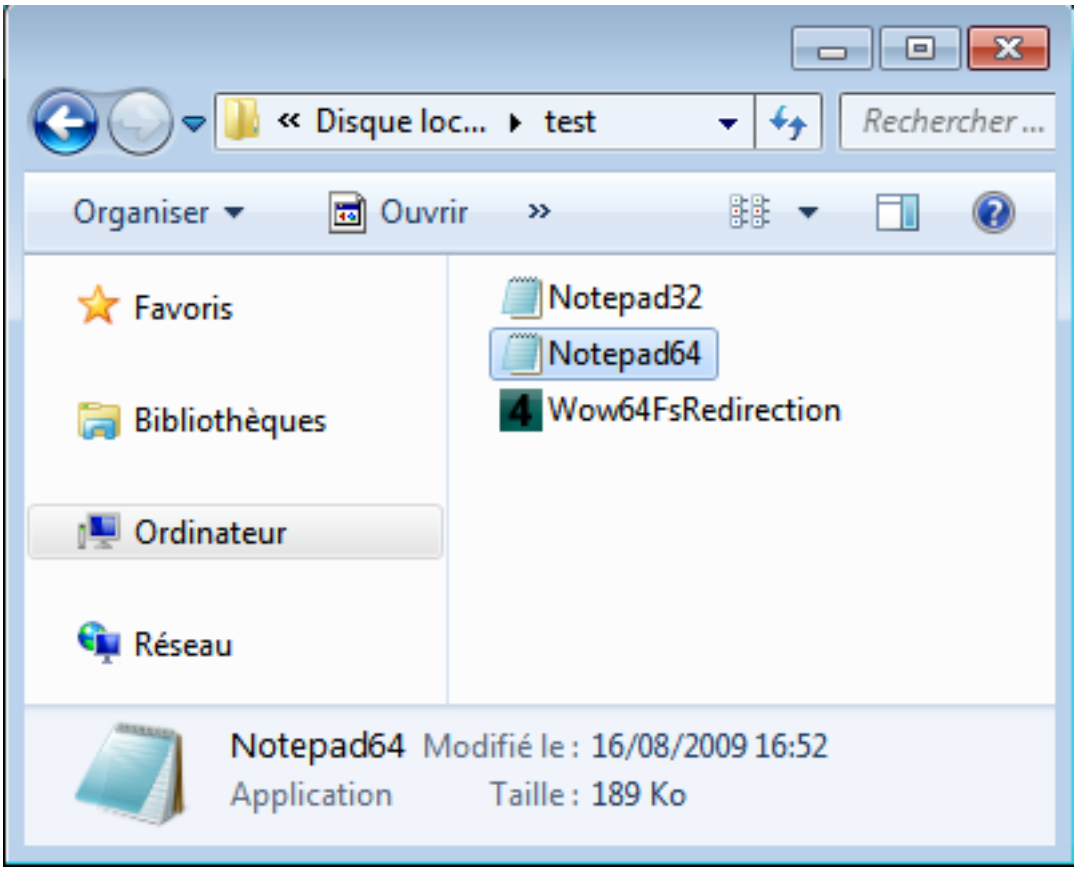


و في الأخير يقوم التطبيق بتفعيل خاصية إعادة التوجيه.
نلاحظ أن المتغير- المخرج - السابق الذي تم حفظه يتم تمريره لدالة تفعيل إعادة التوجيه.

يجب أن ننتبه أن هذه العملية مهمة جدا لكي لا نتسبب في أخطاء تجعل عمل النظام غير مستقر و غير دقيق في تسيير تشغيل تطبيقات 32 بت و 64 بت.

ناتج عملية النسخ، لاحظ الصور التالية:


« Disque loc... ▸ test
Rechercher ...
Organiser
Ouvrir
»
Favoris
Bibliothèques
Ordinateur
Réseau
Notepad32
Notepad64
Wow64FsRedirection
Notepad32 Modifié le : 16/08/2009 16:52
Application Taille : 175 Ko


« Disque loc... ▸ test
Rechercher ...
Organiser
Ouvrir
»
Favoris
Bibliothèques
Ordinateur
Réseau
Notepad32
Notepad64
Wow64FsRedirection
Notepad64 Modifié le : 16/08/2009 16:52
Application Taille : 189 Ko

ملاحظة هامة: إعادة التوجيه يخص فقط تطبيقات 32 بت و لا يخص تطبيقات 64 بت.

الملفات المصدرية مرفقة مع الملف التنفيذي للتجربة، و لا تنسى أخي المتتبع انه يجب عليك تشغيل الملف التنفيذي في بيئة 64 بت، أما بما يخص الملفات المصدرية فيمكنك فتحها و إعادة بنائها على بيئة 32 بت دون مشاكل.

بالتوفيق إن شاء الله

في المقال القادم من السلسلة سوف نتطرق إن شاء الله إلى جعل التطبيق يتعرف على البيئة التي يشتغل فيها، 32 بت أو 64 بت اعتمادا على ما تصدره مكتبات النظام من دوال.

الزرّ و الحمار

(تنبيه: هذه المقالة تجمع بين الجد و الهزل، و يجب أن تقرأ وفق ذلك)

لا أعلم لماذا في دلفي كلما رأيت مكون الزر TButton تخطر على بالي صورة - أكرمكم الله- الحمار، وتتداعي في مخيلتي أوجه الشبه بينهما.

ملاحظة مهمة

قبل أن نسترسل، أود أن أشدد على نقطة هامة جدا، وهي: إني من أشد الناس تقديرا واحتراما وإعجابا بهذا المخلوق، ولن أسمح في مقالتي هذه بأي نوع السخرية أو الاستهزاء تصريحا أو تلميحا، لذلك لنكن واضحين منذ البداية.
وطبعا إحترامي و تقديري وإعجابي هذا ينسحب على ال Button أيضا.

نستأنف موضوعنا
لماذا ذلك .. أي ما هي أوجه الشبه بين ال Button و الحمار ...؟

عندما يبدأ المبرمج مشروعا جديدا في دلفي وتظهر أمامه نافذة النموذج Form1 فإن أول شيء يقوم به هو وضع Button1 على الشاشة ثم تك تك Double Click ويباشر في كتابة التعليمات.

لماذا Button ؟ لأنه الأقرب ليد المبرمج، لا توجد به تعقيدات إضافية كباقي المتحكمات، صغير الحجم لا يأخذ حيزا في الشاشة ، وعند التشغيل يكفيه نخزة في خاصرته.. أقصد click عليها فيقوم بتنفيذ التوليف الذي يحمله مهما بلغ حجمه.

أيضا من المزايا التي يقدمها ال Button هي المنهجية التنظيمية التي بها يمكن للمبرمج أن ينظم تعليماته البرمجية، و يقسمها إلى أجزاء مرنة يمكن إدارتها و السيطرة عليها بسهولة، فمع كل وظيفة أو ميزة جدية يريد أن يضيفها المبرمج لبرنامجه، ما عليه إلا أن يجد حضيرة .. عفوا مرة أخرى.. أقصد نموذج شاشة فيه حيز بسيط ويضع فيه ال Button ثم double click ويضخ فيه ما أراد من كود.

وإذا أراد المبرمج أن يتعقب منطقية برنامجه أو أن يراجع أو يصلح ووظيفة ما فيه، ما عليه إلا أن يذهب مباشرة إلى Button ذات العلاقة و ينبش تحته.

الكثير من المبرمجين يعتمدونه كأفضل وأنسب وأسرع مركوب لتحميل التوليف داخله...

كم من أبطان تحمل داخلها أكواد يصل مداها للعشرات بل و المئات من التعليمات البرمجية.

كم من أبطان تحمل داخلها أكواد مفصلية و جوهرية للبرنامج.

ال Button مطيع، سهل القياد، يتحمل مشاق الكود و ثقله، ولا يشكو مهما حملت عليه.

و هنا جاء الشبه.

و مع هذا ال Button لا يلق ما يستحق من معاملة جيدة.

من جهة القائمين على برنامج دلفي، فمنذ الإصدار الأول من دلفي، لم يجر على TButton أي تغيير أو تطوير، اللهم إمكانية إضافة صورة لها مع نسخة دلفي 2009.

من جهة المبرمجين، مصيبة، فبرغم ما يتحمله ال Button من أعباء، وبرغم ضخامة التوليف المكتوب داخله، فإن حضرة المبرمج لا يتكلف عناء إعطاء إسم محترم له يليق بما يتحمله، فنجده يبقي على الاسم الافتراضي الذي يأتي به كما هو، و كأنه أدنى شأنا من أن يسمى، فنجد أسماء أبطانه: Button1 و الثاني Button2 ثم Button3 وهكذا حمار 1، حمار2 ، حمار3.

أما من جهة أدبيات البرمجيات و تقنياتها ومنهجياتها فإن ال Button يتم تجاهله تماما، برغم عمق تأثيره وتوسع شعبيته، كان من الواجب إفراد منهجية أو مفهوم برمجي خاص به، و يمكن أن يكون إسمه:Button Oriented Programming و إختصارا BOP أي المنحي الزرّي للبرمجة أو البرمجة الزرّية أو البرمجة البُطنية. ويستتبع هذا التحليل بالمنحى البطني، و التصميم البطني و هكذا.

وإذا سألتموني رأيي: فأنا أفضل تسمية هذه المنهجية أو هذا المفهوم بإسم ذو دلالة أكثر، و يستلهم من تراثنا الشعبي العريق، التسمية هي : (خُش يا مبارك بحمارك) .

عموما توجد منهجية بتسمية شبيهة وهي: Cowboy Coding أي البرمجة بطريقة رعاة البقر، ولكن كما تلاحظون تسميتي أفضل و أكثر دلالة.
http://en.wikipedia.org/wiki/Cowboy_coding

ظاهرة ال Button

ظاهرة اعتماد ال Button أو بالأدق إجرائية الحدث الخاص بالنقر على ال Button هي ظاهرة منتشرة عند أغلب المبرمجين. المبتدئون منهم والممارسون المحترفون. وهي ظاهرة مقبولة مبدئيا ، خاصة في بيئة برمجية مرئية سريعة كبيئة تطوير دلفي، فإن كتابة التعليمات البرمجية داخل حدث النقر على الزرّ شيء طبيعي و بديهي، فالمبرمج لن يتكلف عناء إنشاء و تسمية إجرائية جديدة وعناء كتابة أمر الإستدعاء لها؛ بل فقط النقر المزدوج على الزرّ فتقوم دلفي تلقائيا بإنشاء الإجرائية، و أمر الاستدعاء يتم أيضا آليا بمجرد الضغط على الزر بعد تشغيل البرنامج.

المبتدئون يجدون الأمر ممتعا وسهلا، في الواقع يوفر لديهم و قت كبير يستغلونه في التعرّف على باقي نواحي التطوير و البرمجة.

أيضا الممارسون ذوي الخبرة والمحترفون، يجدون الأمر سهلا ومباشرا، إذا أراد المحترف تجربة فكرة ما، أو إعداد نموذج أولي لحل مشكلة برمجية، فإن أسلوب الكتابة داخل الزر هو أيضا أمر طبيعي ومقبول.

أيضا و لحد ما .. قد يكون الأمر مقبولا إذا كان البرنامج صغيرا خفيفا بشاشة واحدة أو اثنتان.

لكن غير ذلك فإن الأمر لايعد مقبولا.

المبتدئ وبعد أخذه فكرة عن الإطار العام للبرمجة و إعداد التطبيقات، يجب أن يتخلص من هذه العادة، وأن ينقل تعليماته البرمجية و يوزعها في إجرائيات خاصة.

و المحترف، بعد أن يختبر تعليماته و يتأكد من صلاحيتها، عليه أن ينقل فورا هذه التعليمات إلى إجرائية خاصة. و لايتكاسل عن هذا الأمر.

الزر أو ال Button كأداة من أدوات واجهة الإستخدام يجب أن لاتختلف وظيفته عن غيره من الأزرار التي نجدها في محيطنا المعاش. زر إنارة المصباح مثلا وظيفته فقط إعطاء الإشارة أو الأمر للمصباح الذي يقوم بالمهام الفعلية لعملية الإنارة. زر بدء تشيغل التكييف يعطي الأمر لجهاز التكييف ليقوم بعمله، أي أن الزر في حد ذاته لا يقوم بمهام الإنارة كما أنه لا يقوم بمهام التكييف.

حديثنا السابق على ال Button ينسحب أيضا على باقي أنواع عناصر واجهة الاستخدام و أنواع الأحداث التي تستقبلها مثل Menu و ListBox و CheckBox وغيرها.

ما ذا بعد ال Button

الآن و بعد أن اتفقنا على ضرورة اعتماد إنشاء الإجرائيات (function, procedure) لوضع تعليماتنا البرمجية فيها؛ ننتقل إلى الحديث عن الإجرائيات.

في بداياتي البرمجية نصحني صديق وأستاذ لي بأن أية إجرائية تبرمجها يجب أن لا تتجاوز تعليماتها البرمجية سبع تعليمات، وعلل بأن العقل البشري أقصى عدد من المعطيات يمكن أن يتصوره أو يتعامل معه في وقت واحد سبع لا أكثر.

الفكرة ليس أن نلتزم بهذا العدد بالظبط، سيكون الأمر صعبا على كثير منا (ولكن ليس مستحيلا)، الفكرة هي أن تكون طول الإجرائيات أصغر ما يمكن، وتحتوي على أقل ما يمكن من معطيات.

وفعلا ، و بالطبع، و كأي نصيحة تكون أكبر من مستواي فإني لم أعمل بها النصيحة الغالية، وأعترف: لقد كنت حـ..... ، أنتم تعرفون.

لقد عانيت ولازلت أعاني نتيجة عدم أخذي بهذه النصيحة، في الكثير من الأحيان إذا ظهرت مشكلة في إحدى برامجي وأحاول أن أتتبعها لإصلاحها أو تعديلها ، أجد أن المنطقة التي بها العلة غالبا ما تكون في إجرائية دسمة طويلة، وعندما أحاول أن أحدد بالظبط أين مكمن الخطأ أو التركيبة المنطقية التي أدت إليه أجد نفسي مرتبكا حائرا أمام إجرائية طويلة وعريضة، تتقدمها صف من المتغيرات variables تزيد عن الثلاثين، ثم مقاطع دسمة كل مقطع لا يخلو من حلقات متداخلة nested loops و جمل شرطية if then كل واحدة منها تؤدي لأخرى، ناهيك عن تعليمات break و exit التي تتكرر و تتوزع على مدى طول الإجرائية.

و النتيجة أن وقتا كبيرا أستغرقه لمحاولة فهم ما تقوم به الإجرائية، ووقتا أكبر لمعرفة مكان العلّة. أحيانا العلة تكون تافهة كنسيان علامة أو استعمال كائن object تم تحريره، لكن لكثرة الأسطر و تشابكها تعجز العين عن رؤية هذه العلة.

طول الإجرائيات ودسامتها هي من أكبر مكامن الثغرات و العلل في البرامج. كما أنها تشكل أكثر العقبات أما تعديل البرنامج أو تطويره أو نقله من بيئة إلى أخرى أو ترجمته من لغة برمجية إلى لغة ثانية. بإختصار: طول الإجرائيات أكبر عائق في سبيل إدارة مرنة للكود.

فكر في البرنامج على أنه مجموعة من الحاويات أو الصناديق، كل إجرائية تمثل صندوقا، ايهما أفضل، أن يكون لديك عشرين صندوقا صغيرا بمتناول حمل اليد الواحدة، و كل صندوق به نوع واحد من المواد، أم أن يكون لديك ثلاثة صناديق كبار كل واحد منها محشور فيه مواد من أنواع مختلفة، و تعجز عن حمله بنفسك.؟

الصناديق الصغيرة مهما كان عددها، سهلة المناولة و يمكنك تنظيمها و إدارتها وتوزيع محتوياتها بالكيفية التي تريدها.

التحكم في طول الإجرائية.. هل هو أمر سهل؟

أعترف، بأن التحكم في طول الإجرائية هو أمر صعب في البداية. عندما تبدأ بالبرمجة وتلك الفكرة المجنونة تسيطر عليك لتنفيذها، و مع تسارع وتيرة الكود، و تلك اللهفة المرتعشة لإضافة خاصية جديدة أو تجربة مسار آخر ثم تشغيل البرنامج لمعرفة النتيجة، و العودة للكود لضبط بعض الأمور أو إضافة استثناء، أو معرفة ماذا لو نقوم بكذا بدلا من كذا، كل هذا يجعل من العقل بعيدا كل البعد عن الاهتمام بتنظيم الكود أو تبسيطه أو تقسيمه إلى إجرائيات. أنت وسط معركة محتدمة، و أصابعك تكاد تقدح شررا ، ولا وقت لديك للتفكير في أمور تنظيمية تنسيقية مملة.

و تسترسل الإجرائية في الطول، وكل تعليمة جديدة تولد عشرة غيرها، خاصة إذا كنت بصدد برمجة ذات علاقة بالرسوميات Graphics أو بالضبط ورفع الكفاءة optimization ؛ فإن الأسطر أمامك تتداعى وتتوالد حتى تصيح الإجرائية بحجم هذه المقالة.

لكن بعد انتهاء المعركة، و اطمئنانك للنتائج، وقبل أن تنتقل إلى موضوع آخر؛ حاول أن تعيد النظر في الإجرائية و قم بتحليلها و تقسيمها إلى إجرائيات أصغر فأصغر.

مع الوقت، تجد نفسك تلقائيا تفكر بطريقة مختلفة عند كتابة الإجرائيات، و بدون عناء منك، يقوم عقلك تلقائيا، بإجبارك على بناء إجرائيات صغيرة خفيفة منذ البداية.

أفضل الممارسات البرمجية

ننتقل إلى بعض أفضل الممارسات والقواعد عند كتابة الإجرائيات:

عدد أقل من الأسطر

حدد عددا معينا من الأسطر في الإجرائية الواحدة ولاتتجاوزه، البعض يقول خمسون سطرا، والبعض يقول عشرون، وآخرون يفضلون خمسة عشر سطرا. المهم أن يكون عدد الأسطر أقل ما يمكن، و مع تزايد مهاراتك البرمجية، و تعود عقلك على هذا القيد: قم بتقليص عدد الأسطر كل مرة.

أسطر قصيرة

لا تجعل سطر التعليمات طويلا. هناك خط عمودي يسار المحرر حاول أن لا تتجاوزه.

مستويان كحد أقصى

لا تجعل من التعليمات تتعمق لأكثر من مستويين كحد أقصى. تجنب الحلقات والجمل الشرطية المتداخلة، التي تبدو مثل السهام الحادة:

```
while do
  while do
    while do
      while do
        while do
          do WhatEverYouWantToDo
        end
      end
    end
  end
end


if
  if
    if
      if
        do something
      end
    end
  end
end
```

التقليل من التعليقات

التعليقات Comments والملاحظات على التعليمات البرمجية أمر مطلوب إذا كانت التعليمة يصعب فهمها أو فهم مهمتها، و لكن من الأساس لماذا نكتب تعليمات صعبة الفهم؟ كلما كانت التعليمات سهلة وواضحة كلما كان ذلك أفضل، و بالتالي نستغني على التعليقات. الإجرائية التي لا تحتاج إلى تعليق أفضل من تلك التي تحتاجها.

لا تعيد نفس الكود مرتين

إذا وجدت نفسك تنسخ تعليمات من إجرائية أخرى وتعيدها، فهذا مؤشر لأن تعيد التفكير في التعليمات المنسوخة وأن تضعها في إجرائية خاصة بها.

قم بحوسبة الإجرائية في عقلك أولا

قبل أن تقوم بالأمر Compile لبرنامجك للتأكد من سلامة الكود المكتوب، حاول أن تقوم بهذه العملية في عقلك أولا. راجع ببصرك التعليمات المكتوبة وتتبعا سطرا سطرا وحاول أن تكتشف العلل بنفسك قبل أن يكتشفها ال compiler . مع الممارسة الدائمة لهذه العادة سيتعلم عقلك كيف يتجنب مكامن الأخطاء التي عادة ما تقع بها.

التسمية الواضحة والقصيرة

تسمية الإجرائية يجب أن يكون دالا على ما تقوم به، وأن لا يكون طويلا، و أن لا يكون مختصرا في حرفين أو ثلاث. أفضل التسميات ما كان يحوي على كلمة واحدة أو كلمتين كحد أقصى.

يقول الخبراء إذا كانت التسمية مختصرة جدا أو طويلة جدا فقد يعني هذا أن الاجرائية تقوم بعدة أمور مما يصعب عليك إيجاد إسم واضح لها بكلمة أو اثنتين. وهذا يقودنا إلى القاعدة المهمة التالية.

مهمة واحدة لكل إجرائية

الإجرائية يجب أن تقوم بمهمة واحدة فقط وليس أكثر من واحدة. إذا لا حظت أن التعليمات استرسلت فقد يعني هذا أن مهام أخرى بدأت بالظهور ، ووجب إفراد إجرائية منفصلة خاصة بها.

هذه ربما أهم ممارسة يجب أن تمتلك مهارتها. إذا حققت هذا الأمر، فستجد أن باقي النقاط التي سردناها تتحقق تلقائيا.

المراجعة و التنقيح

بين كل حين و آخر قم بمراجعة الكود، وطبق عليه ماسبق من قواعد، و أجعله في كل مرة أكثر بساطة ووضوحا.

خطوة خطوة الجزء الثاني

نواصل ما بدأناه من جولات في المقالة السابقة و نتعرف أكثر على مفاهيم المنحى للكائن.

الجولة الخامسة

سوف نقوم الآن بإنشاء صنفية class جديدة، لكن هذه المرة ستكون مشتقة من صنفية TPerson ، الصنفية الجديدة نريد بها تمثيل فئة معينة من الأفراد وهي فئة الموظفين، و سنسمي الصنفية الجديدة TEmployee ، سنقوم بتعريف هذه الصنفية في وحدة uOOP تحت تعريف صنفية TPerson وذلك كالتالي:

```
TPerson = class(TObject)
-------
-------
end;

TEmployee = class(TPerson);
```

كما هو واضح أعلاه؛ قمنا فقط بتعريف الصنفية الجديدة وبيان أنها مشتقة من TPerson . ولم نضع فيها أي عنصر آخر.

نجرب الصنفية TEmployee، وأقترح هنا إنشاء زر آخر Button2 لتجربة هذه الصنفية. ستكون التعليمات مطابقة تماما لتلك التي داخل Button1 ، التغيير الوحيد هو استخدام TEmployee بدلا من TPerson وتسمية المتغير بإسم Emp كالتالي:

```
procedure TForm1.Button2Click(Sender: TObject);
var
  Emp: TEmployee;
begin

  Emp := TEmployee.Create;

  try
    Emp.FirstName := 'Ahmad';
    Emp.LastName := 'Hamza';
    Emp.BirthDate := EncodeDate(1980, 3, 15);

    ShowPerson(Emp);

  finally
    Emp.Free;
  end;

end;
```

نقوم بتجربة البرنامج، و ونشغل الزر Button2 و سنرى أن النتيجة مطابقة لنتيجة عمل الزر الأول.

لماذا هذا التطابق برغم أن تعريف الصنفية TEmployee جاء خاليا من أية عناصر أو إجرائيات. لأن الصنفية TEmployee مشتقة من صنفية TPerson وبالتالي سترث جميع صفاتها وخصائصها. لهذا يمكن لأي معرف أو متغير من نوع الصنفية الجديد (المتغير Emp في مثالنا) أن ينفذ لخصائص الصنفية الأصل مثل FirstNam أو GetAge.

الأمر الآخر الملفت للنظر؛ هو أننا قمنا بنداء نفس الإجرائية ShowPerson ، فبرغم أن هذه الإجرائية في تعريفها تستقبل معطى من نوع TPerson إلا أنها لم تمانع عندما مررنا لها معطى من نوع TEmployee، لماذا؟ لأنه بالنسبة لها نوع TEmployee يحتوي ضمنا على النوع TPerson.

لمزيد من الاستكشاف يمكننا زرع الكود التالي في بداية تنفيذ الإجرائية:

```
procedure TForm1.ShowPerson(P: TPerson);
begin

  if P is TPerson then Caption := 'Person';
  if P is TEmployee then Caption := Caption + ' Employee';
  Caption := Caption + ' ' + P.ClassName;
  . . . .
```

ونستدعي الاجرائية بواسطة Button1 ثم بواسطة Button2 ونلاحظ ثم نستنتج الفرق.

إضافة خاصية جديدة

نضيف الآن خاصية property جديدة لصنفية TEmployee تمثل المرتب. الخاصية إسمها Salary

كالتالي:

```
  TEmployee = class(TPerson)
  private
    FSalary: integer;
    procedure SetSalary(const Value: integer);
  public
    property Salary: integer read FSalary write SetSalary;
  end;
```

لنلاحظ أن هذه الخاصية الجيدة تخص فقط الصنفية TEmployee و لا علاقة لها بالصنفية الأصلية TPerson، أي أن TEmployee لديها جميع صفات TPerson بالوراثة ثم تزيد عليها الخاصية Salary.

لنرى الآن كيف نتفاعل مع هذه الخاصية الجديدة. في الحدث الخاص بالزر Button2 نستخدم خذه الخاصة ونسند لها قيمة:

```
procedure TForm1.Button2Click(Sender: TObject);
 . . . . . .
 . . . . . .

    Emp.LastName := 'Hamza';
    Emp.BirthDate := EncodeDate(1980, 3, 15);
    Emp.Salary := 1200;
    ShowPerson(Emp);
 . . . . . .
 . . . . . .
end;
```

ثم في الإجرائية ShowPerson نحاول إظهار قيمة هذه الخاصية على الشاشة ضمن القيم الأخرى.

للتذكير تعليمات الإظهار في إجرائية ShowPerson كالتالي:

```
  Canvas.TextOut(10, 10, P.FirstName);
  Canvas.TextOut(10, 30, P.LastName);
  Canvas.TextOut(10, 50, DateToStr(P.BirthDate));

  Canvas.TextOut(10, 80, P.GetFullName);
  Canvas.TextOut(10, 100, IntToStr(P.Age));
```

الآن لو أضفنا التعليمة :

```
Canvas.TextOut(10, 120, IntToStr(P.Salary);
```

فإن المترجم سيعطينا رسالة خطأ بأن Salary غير معرفة (Undeclared identifier). وهو هنا يقصد الخاصية Salary عندما جاءت بالسياق مع المتغير P أي P.Salary. ولنفهم لماذا فإن P من نوع الصنفية TPerson والتي هي في الأصل لا تحوي الخاصية Salary.

إذا لماذا وافقت الإجرائية على استقبال القيمة في المعطى P رغم أن القيمة من نوع TEmployee ؟ كما ذكرنا سابقا: لأنه بالنسبة لها فإن نوع TEmployee يحتوي ضمنا على النوع TPerson .

إذا المترجم وافق على إستقبال قيمة من نوع TEmployee ما دام هذا النوع مشتق من TPerson لكنه لم يوافق على مناداة الخاصية Salary بالسياق مع P لأن P معرف في الأصل على أنه من نوع TPerson الذي لايعرف هذه الخاصية.

ما الحل؟

الحل باستخدام تقنية الصبّ أو القولبة Casting. وهو أن نصب النوع TPerson في قالب من نوع TEmployee كالتالي: TEmployee(P).Salary أي أننا نلبس P ثياب TEmployee.

هنا أجبرنا المترجم على أن يعامل P على أنه من نوع TEmployee ولأن TEmployee مشتق من TPerson فإنه لا يمانع أن يحتضن P التي هي من نوع TPerson .

لذلك سنعدل من تعليمة الإظهار لتكون كالتالي:

```
Canvas.TextOut(10, 120, IntToStr(TEmployee(P).Salary));
```

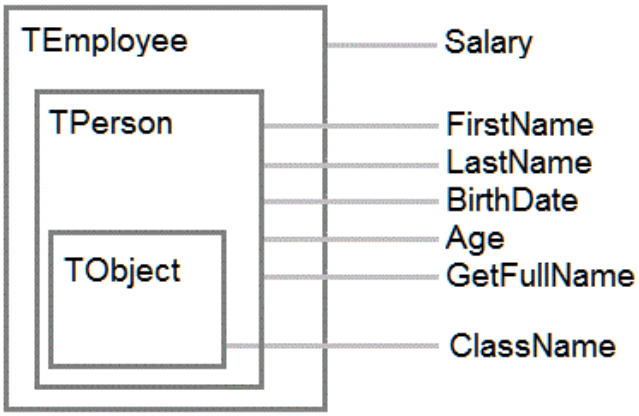
ولأن الخاصية Salary موجودة فقط في النوع TEmployee ؛ نقوم بإضافة جملة شرطية تتأكد من أن قيمة P من نوع TEmployee .

```
if P is TEmployee then
  Canvas.TextOut(10, 120, IntToStr(TEmployee(P).Salary));
```

ملاحظة: في دلفي يمكن أيضا أن نقوم بعملية الصب (Casting، القولبة، التلبيس) بالطريقة التالية:

```
(P as TEmployee).Salary
```

والأمر متروك لتفضيل المبرمج.



التحوّر والتشكّل Polymorphism

في أعراف المنحى للكائن نفس الدالة function أو المنهاج/الإجرائية method يمكن أن تنفذ باكثر من طريقة عبر سلسلة الصنفيات المشتقة من بعض.

في صنفية TPerson توجد الدالة GetFullName التي تقوم بجمع الإسم الأول مع الثاني.
يمكننا أن نجعل من الصنفية TEmployee تقوم بإظهار الإسم بصورة مختلفة مع الحفاظ على نفس إسم الدالة.

لتنفيذ ذلك نقوم بالتالي:

أولا نقوم بإضافة وسم virtual في تعريف الدالة GetFullName في صنفية TPerson:

```
  function GetFullName: string; virtual;
```

هذا الوسم يتيح للصنفيات المشتقة أن تعيد تعريف نفس الدالة وأن ترث تنفيذها إذا أرادت.

ثانيا، نعيد تعريف نفس الدالة في صنفية TEmployee بالطريقة التالية:

```
TEmployee = class(TPerson)
.. .. ..
public
  function GetFullName: string; override;
.. .. ..
```

لاحظ الوسم override في نهاية تعريف الدالة، والذي يشير إلى أن الدلة في هذه الصنفية ستستولي على التنفيذ السابق في الدالة المشتقة.

في جسم الدالة نقوم بإدخال التعليمات الجديدة الخاصة بإظهار الاسم، وهنا لنا الخيار بأن نقطع العلاقة مع التنفيذ السابق ونحدد تنفيذا جديدا مثل:

```
function TEmployee.GetFullName: string;
begin
  result := 'Mr.' + ' ' + LastName + ', ' + FirstName;
end;
```

أو نستخدم التنفيذ السابق في صنفية TPerson مع بعض التحوير:

```
function TEmployee.GetFullName: string;
begin
  result := 'Mr. ' + inherited GetFullName;
end;
```

لاحظ الأمر inherited والذي يعني تنفيذ ما هو موروث.

لنختر أيا من التنفيذين ونجرب نتيجته، وسنلاحظ اختلاف عرض الاسم الكامل بحسب سياق النوع، بحيث لو تم استدعاء هذا الدالة بالسياق مع النوع TPerson تعرض الإسم بصيغة تختلف عن ما لو تم استدعاؤها بالسياق مع TEmployee.

من الصنفية الواحدة يمكننا اشتقاق أكثر من صنفية، فكما فعلنا مع صنفية TPerson و اشتققنا منها صنفية TEmploye ؛ يمكننا أيضا اشتقاق صنفية أخرى مثل TChild .

من الصنفية المشتقة يمكن أن نشتق صنفية أخرى. مثلا صنفية TEmployee نشتق منها صنفية TManager.

سلسلة الاشتقاقات غير محدودة العدد وكل واحدة ترث خصائص وسلوكيات ما قبلها.

الجولة السادسة

في هذه الجولة نحاول أن نقترب قليلا و نتفهم طبيعة المتغيرات التي من نوع كائني. نحن في أمثلتنا السابقة استخدمنا المتغير P و المتغير Emp ، الأول من نوع TPerson و الثاني من نوع TEmployee وكلاهما تعود جذورهما إلى النوع الأصلي TObject.

المتغير من نوع كائني يختلف عن المتغيرات من الأنواع الأساسية مثل integer و string كونه لا يحمل قيمة النوع في حد ذاته.

لنوضح أكثر؛ عندما نعرّف المتغير من نوع A على أنه integer ونسند له قيمة 10 فإن المتغير يكون حاملا لهذه القيمة (قسم في الذاكرة يحوي قيمة A) . تابع معي

```
var
  A, B, C: integer;
begin
  A := 10;
  B := A;          // 10
  A := A + 5;      // 15
```

المتغير B خصص له قسم آخر في الذاكرة وأعطيناه نفس قيمة A ، وعند أضافة 5 لقيمة A فإن A ستصبح قيمتها 15 بينما B لا تزال قيمتها ثابتة.

```
C := A + B;       // 25
```

المتغير C له قسم خاص به في الذاكرة، و أصبحت قيمته الآن مجموع A و B أي 25

المتغيرات الثلاث كل واحد منهم مستقل عن الآخر ويحمل قيمه الخاصة به في قسم خاص به في الذاكرة.

لو طبقنا نفس السيناريو على المتغيرات من نوع كائني:

```
var
  oA, oB, oC: TPerson;
begin

  oA := TPerson.Create;
  oB := oA;
  oC := oA;

  oA.FirstName := 'Ahmad';
  oB.LastName := 'Hamza';

  Caption := oC.GetFullName;    // 'Ahmad Hamza'
  oA.Free;
  oA := nil;
  oB := nil;
  oC := nil;
```

عندما جسدنا كائنا من نوع TPerson أصبح لهذا الكائن وجود في مكان ما في الذاكرة.

المتغير oA له مكان آخر في الذاكرة أيضا ولكنه لا يحوي الكائن وإنما يحوى عنوان موقع الكائن في الذاكرة أي أنه مؤشر Pointer إلى موقع الكائن في الذاكرة.

المتغير oB قبل أن نسند له أية قيمة؛ له موقع في الذاكرة ولكن لايحوي أية قيمة ذات معنى؛ أو بالأصح يحوي قيمة عشوائية.

بعد oB := oA ، تم اسناد قيمة oA إلى المتغير oB أي أنه الآن يحوي نفس قيمة oA وهو عنوان في الذاكرة أي أنه يشير إلى نفس الكائن الذي يشير إليه oA ، وكذبك الأمر بالنسبة للمتغير oC .

أي أن المتغيرات oA و oB و oC يشيرون الآن إلى نفس الكائن.

وبهذا يمكننا أن ننفذ للكائن من خلال المتغير oA مثلا ونعطي قيمة للخاصية FirstName ("Ahmad")، و أيضا ننفذ لنفس الكائن من خلال المتغير oB ونعطي قيمة لخاصيته LastName ("Hamza")، لذلك من المنطقي تبعا لذلك لو استفسرنا عن قيمة GetFullName من خلال oC فسيعطينا: "Ahmad Hamza".

بعد الإنتهاء من تعاملنا مع الكائن تم تحريره أو إنهاؤه. من خلال الأمر oA.Free.

طبعا نحن نعلم الآن لماذا لم نتبع الأمر السابق بأمر oB.Free و oC.Free ؟

بعد تحرير الكائن قمنا بالتأكيد على أن المتغيرات الآن لا تشير إلى أي شيء وذلك بتخصيص قيمة nil لكل منها.

يمكن أيضا تحرير الكائن و تصفير المتغير في وقت واحد عن طريق الإجرائية FreeAndNil() .

```
FreeAndNil(oA);
```

هذه الإجرائية تقوم بتحرير الكائن الذي يشير إليه المتغير oA ثم تقوم بإعطاء قيمة nil للمتغير.

نصائح عند التعامل مع الكائنات

- يجب التعامل مع الكائنات بانتباه وجدية.
- ضرورة إنهاء الكائن وإفناؤه بمجرد انتهاء الحاجة إليه.
- ضرورة تصفير أي متغير يشير إلى هذا الكائن.
- أن تكون برامجك والتعليمات فيها أكثر هيكلة وتنظيما.

مزالق يجب الإنتباه إليها

ملاحظة: القسم التالي يتناول طبيعة التعامل مع الكائنات بتفصيل أكثر، فإذا وجدت صعوبة في تتبعه الآن؛ يمكنك تخطّيه والانتقال مباشرة إلى الجولةالتالية.

كما أشرنا؛ يجب التعامل مع المتغيرات من النوع الكائني بشيء من الانتباه ، و أن نعوّد أنفسنا على آلية عملها حتى لا نقع في مطباتها.

فيما يلي سنتتبع سلوك المتغيرات ونفحص قيمها عن قرب.

```
var
  oA: TPerson;
begin
```

تم تعريف المتغير oA من نوع TPerson ، هذه المتغير لايشير إلى أي كائن.

لكنه قد يحوي قيمة عشوائية لا معنى لها في سياق برنامجنا. ويمكننا معرفة هذه القيمة من خلال التعليمة Pointer(oA) .

```
oA := TPerson.Create;
```

تم إنشاء كائن من نوع TPerson و في نفس الوقت تم تخصيص قيمة للمتغير oA هذه القيمة هي مؤشر للكائن الذي تم إنشاؤه.

ماهي قيمة oA الآن؟ هي رقم يشير إلى عنوان موقع في الذاكرة. ولمعرفة هذا الرقم نستخدم التعليمة Pointer(oA) (من خلال شاشة Evaluate/Modify مثلا)

```
  oA.FirstName := 'Ahmad';
  oA.LastName := 'Hamza';
```

من خلال المتغير oA نفذنا إلى خاصية في الكائن وهي FirstName وأعطيناها قيمة : "Ahmad". وأعطينا قيمة للخاصية LastName : "Hamza" . إذا إلى هذا الحد تم التعامل مع الكائن وإعطاء قيم لخاصيتين فيه.

```
  oA.Free;
```

هنا استخدمنا المتغير oA للنفاذ للكائن وطلبنا منه تدمير نفسه وإنهاء وجوده. الكائن الآن لاوجود له في الذاكرة. ولو حاولنا مخاطبة الكائن مرة أخرى من خلال المتغير oA ؛ فسينتج عن ذلك خطأ في وقت التشغيل. لأن الكائن لم يعد له وجود.

السؤال هل المتغير oA تغيرت قيمته بعد تحرير الكائن بحيث لايشير إلى شيء؟

لو تفحصنا قيمة المتغير oA (Pointer(oA)) سنجدها ثابتة لم تتغير. أي لا تزال تحمل قيمة العنوان القديم للكائن الذي لم يعد له وجود!

هل هذه مشكلة؟

نعم.

فلنتخيل مثلا أن كائنا ما قد تم تحريره و إنهاؤه في مكان ما في البرنامج، ثم يقوم المبرمج في مكان آخر من البرنامج بمحاولة التعامل مع هذا الكائن، عادة يلجأ المبرمج إلى فحص قيمة المتغير أولا والتأكد من وجود قيمة له قبل التعامل مع الكائن وذلك عن طريق الدالة Assigned() :

```
if Assigned(oA) then . . .
```

أو عن طريق الاستفسار إذا كان محتواها لا يساوي nil:

```
if oA <> nil then . . .
if Pointer(oA) <> nil then . . .
```

التعليمات السابقة كلها سوف ترد بالإيجاب! فينخدع المبرمج جرّاء ذلك، ويظن أن الكائن لايزال حيا في الذاكرة، و يبدأ في التعامل مع هذا الكائن المرحوم، وهنا تقع المشاكل.

هل توجد طريقة مباشرة نتأكد بها من أن المتغير يمثل فعلا كائنا قائما في الذاكرة؟

إلى حدّ علمي لا توجد!

الإجراء المناسب للمبرمج هو أن يتخذ سياسة دفاعية بأن يقوم بتصفير المتغير بعد تحرير وإنهاء الكائن.

لذلك نكرر بأنه يجب الإنتباه عند التعامل مع الكائنات، و أن يكون حاضرا في ذهننا دائما ضرورة إنهاء الكائن بعد انتفاء الحاجة إليه، بل يستحسن أن نقوم بكتابة تعليمات إنهاء الكائن فور كتابة تعليمات الإنشاء، وقبل كتابة أية تعليمات تتعامل معه.

انتهت الجولة!

DELPHI4ARAB

منتدى دلفي للعرب منكم و إليكم

ساهم في تطويره بمشاركتك في المنتدى و في مجلة منتدى دلفي للعرب

لمشاركتك في مقالات المجلة، أرسل فقط المقالة بأحد الصيغ **Doc/Docx/ODF** دون تنسيق مسبق إلى إدارة المنتدى delphi4arab@gmail.com