الجزء الثاني

إطار عمل .NET Framework

PDF Factory Pro trial version www.fastio.com

الفئات الأساسية

تحتوي مكتبة فئات إطار عمل NET Framework. على مئات الفئات التي تقوم بالعديد من الوظائف المختلفة والمتنوعة، واستخدامها يعتبر جزء لا يتجزأ من مراحل بناء برامجك ومشاريعك بـــ Visual Basic .NET. ولكن الإلمام بجميع هذه الفئات يحتاج إلى وقت طويل جدا من عمرك البرمجي، ولا أذيع سرا إن أخبرتك أني حتى لحظة كتابة الصفحة الأخيرة من هذا الكتاب مازلت اكتشف المزيد والمزيد من هذه الفئات وأعضائها.

يقدم لك هذا الفصل مدخلاً إلى الفئـــات الأساســـية لإطـــار عمـــل NET Framework.، وسنستمر في التوغل في معظم باقي الفئات الأخرى بالتدريج في الفصول اللاحقة حتى نهاية هذا الكتاب، وعلي أن أذكرك هنا بان هذا الكتاب ليس مرجع من مراجع MSDN، فلا تتوقـــع منـــي القيام بشرح جميع مكتبة فئات NET Framework. وأعضائها بنفسي. لذلك، عليك البحث عـــن كافة تفاصيل الفئات وأعضائها في مستندات NET Documentation.

الفئة System.Object

جميع فئات إطار عمل NET Framework. وجميع التركيبات والأنواع الأخرى مشتقة وراثيا – بشكل مباشر او غير مباشر - من الفئة System.Object، مما يعني ان جميع الأعضاء المعرفـــة في الفئة System.Object ستكون موجودة أيضا في أنواع البيانات التي تنشئها. المزيد أيـــضا، يمكنك إسناد أي قيمة إلى المؤشرات من النوع Object (فهي الفئة القاعدية Base Class لـــذلك تستطيع إسناد أي قيمة من فئة مشتقة Derived Class إلى فئة قاعدية كما وضحت لك في الفصل الرابع **الوراثة**):

```
' يمكنك استخدام الصيغة المختصرة
' أيضا Dim X As Object
Dim X As System.Object

X = New AnyClass
```

الشيء الوحيد الذي لا يشتق وراثيا من الفئة System.Object في عالم .NET. هي الواجهات Interfaces والتي ناقشناها في الفصل السابق **الواجهات، التفويض، والمواصفات**.

طرق الفئة Object

بما ان جميع البيانات مشتقة من الفئة Object، فمن الجيد الإلمام بجميع الطرق التابعة لهذه الفئة حيث انها ستكون موجودة مع جميع البيانات الأخرى.

الطريقة Equals():

هذه الطريقة قابلة لإعادة القيادة Overridable و تعود بالقيمة True ان كان المؤشر المرسل يشير إلى نفس الكائن الحالي:

```
Dim x As New TestClass()
Dim y As Object = x

ArabicConsole.WriteLine(y.Equals(x)) ' True
```

مع ذلك، فإن معظم الفئات والأنواع الأخرى من البيانات في إطار عمل .NET Framework. تقوم باعادة قيادة Overrides هذه الطريقة (وهذا ما ستقوم به انت في العادة). فمثلا، المتغيرات من النوع ذو القيمة Value Type Variables تقوم بإعادة قيادة هذه الطريقة بحيث تعود بالقيمة True ان كانت قيمة المتغير المرسل تساوي قيمة المتغير الحالي:

```
Dim X As Integer = 10

ArabicConsole.WriteLine(X.Equals(10)) ' True
```

الطريقة GetType():

تعود هذه الطريقة بقيمة من النوع System.Type تمثل نوع الكائن الحالي.

انظر أيضا

سأتحدث عن الفئات من النوع System.Type لاحقا في الفصل الثاني عشر **فئات الانعكاس** Reflection Classes.

PDF

الطريقة ()ToString:

هذه الطريقة قابلة لإعادة القيادة Overridable وهي تعود بقيمة حرفية (من النوع String) تمثل الاسم الكامل للفئة التي أنشئ منها الكائن الحالي (بما في ذلك مجالات الأسماء Namespaces):

```
Dim obj As New TestClass()

ArabicConsole.WriteLine(obj.ToString) ' MyNamespace.TestClass
```

مع ذلك، فمعظم فئات إطار عمل NET Framework. تقوم بإعادة قيادة هذه الطريقة بحيث تعود بالقيمة التي يحتويها الكائن:

```
Dim X As Integer = 10
Dim Y As String = "xxx"

ArabicConsole.WriteLine(X.ToString) ' 10
ArabicConsole.WriteLine(Y.ToString) ' xxx
```

الطريقة ()ReferenceEquals:

تعود الطريقة ()ReferenceEquals بالقيمة True ان كان المؤشران المرسلان يشيران إلى نفس الكائن، وبعبارة أخرى، تعمل الطريقة ()ReferenceEquals عمل المعامل Is:

```
Dim Turki As New Person()
Dim Khaled As Person = Turki

ArabicConsole.WriteLine(Turki Is Khaled) ' True
ArabicConsole.WriteLine(Person.ReferenceEquals(Turki, Khaled)) ' True
```

الطريقة ()ReferenceEquals مشابهة للطريقة ()Equals المذكورة سابقاً، الا انها تختلف عنها في نقطتين: الأولى ان الطريقة ()ReferenceEquals طريقة مشتركة Shared Method و النقطة الثانية انها غير قابلة لإعادة القيادة Not Overridable .

الطريقة ()MemberwiseClone:

تستخدم الطريقة ()MemberwiseClone لنسخ الكائن وانشاء نسخة جديدة طبق الاصل منه، وقد ذكرت مثالا يستخدمها في الفصل الخامس **الواجهات، التفويض، والمواصفات** وبالتحديد عند شرح **الواجهة ICloneable**. من المهم ان اذكر هنا ان محدد الوصول للطريقة

()MemberwiseClone هو Protected، وعليه، فلن تتمكن من الوصول لها إلا من داخل الفئة نفسها (فهي مشتقة تلقائيا من الفئة System.Object).

انظر أيضا

لمزيد من التفاصيل حول محدد الوصول Protected ومحددات الوصول الأخرى، راجع الفصل الرابع **الوراثة**.

الطريقة ()Finalize:

الطريقة ()Finalize قابلة لإعادة القيادة Overridable وهي طريقة محمية Protected أيضا ولن تتمكن من استدعائها إلا من داخل الفئة –حالها كحال الطريقة السابقة ()MemberwiseClone. تحدثت عن هذه الطريقة سابقا في الفصل الثالث **الفئات والكائنات**، وذكرت انها تعمل عمل المهدمات Destructors.

البيانات المرجعية والبيانات ذات القيمة مرة أخرى

اتفقنا سابقا على ان جميع البيانات في إطار عمل NET Framework. تنقسم إلى قسمين هما بيانات مرجعية Reference Type وبيانات ذات قيمة Value Type، واتفقنا أيضا على ان البيانات المرجعية يتم حفظها في قسم خاص لها من الذاكرة يسمى Managed Heap، وذكرت كل التفاصيل المتعلقة بالبيانات المرجعية سابقا في الفصل الرابع **الفئات والكائنات**.

الاعداد Numbers (سواء كانت صحيحة او عشرية)، القيم المنطقية Booleans، التركيبات من النوع Structures او Enums جميعها بيانات من النوع ذو القيمة Value Type، وعندما اطلق عليها كلمة فئات عددية او فئات من النوع Boolean –في هذا الفصل في التحديد- فلا اعني انها فئات حقيقية كما تعرفها بداخل التركيب Class ... End Class، حيث ان الفئات التي تعرفها بنفسك هي من النوع المرجعي Reference Type، بينما الفئات السابق ذكرها من النوع ذو القيمة Value Type.

الفئات من النوع ذو القيمة مشتقة وراثيا –وبشكل تلقائي- من الفئة System.ValueType، حيث انها تحتوي على طرق متشابهة في الاداء (مثل الطريقة ()Equals والتي تم إعادة قيادتها Overrides بحيث تقارن القيم التي تحملها المتغيرات وليس الكائنات التي تشير لها المؤشرات).

معظم فئات إطار عمل NET Framework. من النوع المرجعي Reference Type، إلى جانب الفئات التي تعرفها بنفسك، والمصفوفات Arrays، والحروف Strings. بينما الفئات او جميع البيانات العددية Numbers، التركيبات من النوع Structures او Enums جميعها من الأنواع ذات القيمة Value Type. المزيد أيضا، يمكنك معرفة ما اذا كانت فئة معينة من النوع المرجعي Reference Type او ذو القيمة Value Type بقراءة مستندات NET. Documentation او الانتقال إلى نافذة مستعرض الكائنات Object Browser (**شكل 1-9 صفحة 23**) حيث تعطي رموز مختلفة للفئات والتركيبات.

بصفة عامة، المتغيرات ذات القيمة Value Type أسرع بكثير من المتغيرات المرجعية Reference Type واقل استهلاكا للذاكرة لسببين: الأول ان بيانات المتغيرات ذات القيمة تصل اليها مباشرة عن طريق المتغير نفسه بينما بيانات المتغيرات المرجعية فالوصول اليها يكون بالمرور أولا بالمؤشر (المتمثل في المتغير نفسه) لتصل إلى بيانات الكائن الموجود في القسم Managed Heap من الذاكرة. السبب الثاني ان عملية تفريغ محتويات البيانات من الذاكرة لا تتطلب استخدام الـ Garbage Collection فهي مخصصة للقسم Managed Heap أي للبيانات المرجعية. لذلك، يفضل دائما استخدام المتغيرات ذات القيمة Value Type في برامجك، والطريقة الوحيدة التي تمكنك من تعريف أنواع ذات قيمة Value Type جديدة في NET. Visual Basic هي بتعريف تركيبات من نوع Structures مع العلم بأنك لن تستطيع تطبيق مبدأ الوراثة Inheritance مع هذه التركيبات.

مع ذلك، توجد حالات معينة تجعل المتغيرات المرجعية أسرع من المتغيرات ذات القيمة، مثلا عند عملية إسناد القيمة، فلو كان لدينا التركيب التالي:

```
Structure Person
    Public Name As String
    Public Age As Integer
End Structure
```

فان عملية إسناد القيم بين المتغيرات من نوع هذا التركيب ستكون أبطأ بكثير فيما لو كان التركيب السابق من النوع المرجعي (أي عرف باستخدام Class ... End Class)، حيث ان عملية إسناد القيم بين المتغيرات ذات القيمة تؤدي إلى نسخ **جميع محتويات وعناصر** التركيب التابع له المتغير، بينما نجد في المتغيرات المرجعية ان إسناد القيم يؤدي إلى نسخ **المؤشرات** فقط (حجمها لا يتجاوز 4 بايت).

PDF

الصندقة واللاصندقة

حالة أخرى تجعل المتغيرات المرجعية أسرع من المتغيرات ذات القيمة تعرف **بالصندقة Boxing واللاصندقة Unboxing**. تحدث الصندقة نتيجة إسناد قيمة لمتغير من نوع ذو قيمة Value Type إلى متغير من النوع Object، او ارسال متغير من النوع ذو القيمة إلى اجراء يستقبل وسيطة من النوع Object:

```
Dim X As Integer = 10

' Boxing الصندقة
Dim Y As Object = X
```

عملية الصندقة تؤدي إلى إنشاء متغير مخفي من النوع المرجعي Reference Type، يستخدمه المتغير Y السابق في كل مرة تود الوصول إلى القيمة الفعلية التي يحتويها المتغير. وان قمت بتطبيق العكس (إسناد قيمة متغير من النوع Object إلى متغير ذات قيمة Value Type)، فانك ستحدث ما يعرف باللاصندقة Unboxing:

```
' بافتراض ان العبارة Option Strict On مفعلة
' لذلك استخدمت الدالة CInt
X = CInt(Y)
```

الصندقة واللاصندقة تستهلك وقت إضافي في عملية إسناد القيم، او حتى إرسال القيم إلى وسيطات من النوع Object إلى الإجراءات (كالطريقة WriteLine() التابع للكائن ArabicConsole). خلاصة القول، استخدم البيانات ذات القيمة Value Type دائما ان كنت لا ترغب في تطوير النوع باستخدام الوراثة ولا تتوقع عمليات إسناد قيم متعددة او حدوث أي عمليات صندقة او لاصندقة أخرى.

الفئات الحرفية

عند التصريح عن متغير حرفي جديد من النوع String عليك إسناد قيمة له قبل الوصول إلى أعضائه، فالشيفرة التالية ستظهر رسالة خطأ:

```
Dim x As String

' رسالة خطأ
ArabicConsole.WriteLine(x.Length)
```

PDF Factory Pro trial version www.fineprint.com

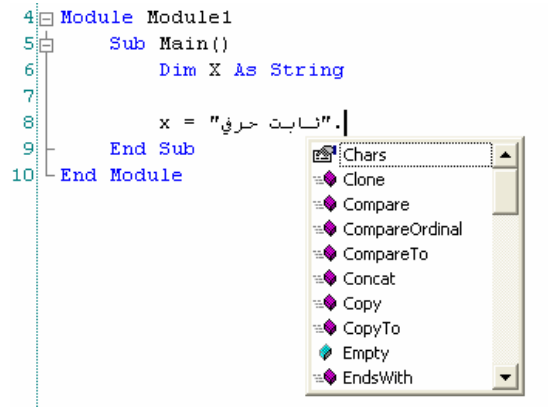
وذلك لاننا لم ننشئ نسخة من الكائن x قبل استخدام الخاصية Length التابعة له، ولانشاء نسخة جديدة من الكائن الحرفي استخدم الكلمة المحجوزة New او اسند قيمة ابتدائية للمتغير لحظة التصريح عنه:

```
' String الوسيطة الأولى تستقبل قيمة من النوع Char وليس
Dim X As New String("A"c, 5)

Dim Y As String = "AAAAA"
```

اعلم جيدا مدى الارتباك التي قد تسببه لك فكرة تحول المتغيرات الحرفية من سلاسل حروف –في لغات البرمجة السابقة- إلى كائنات منشأة من فئات، ولكن عليك من الآن وصاعداً التأقلم مع هذا الأمر حتى لا تسبب في ايجاد الشوائب في برنامجك. وبما أن المتغيرات الحرفية من النوع المرجعي Reference Type، فان عملية إسناد القيم بين المتغيرات الحرفية لا تؤدي إلا لنسخ مؤشرات الكائنات فقط:

```
Dim X As String = "قيمة حرفية"
Dim Y As String = X

' اثبات ان كلا المؤشرين X و Y
' يشيران إلى نفس الكائن
ArabicConsole.WriteLine(X Is Y) ' True
```

قد اسبب لك إرباكا أكثر إن زدتك من الشعر بيت وأخبرتك انه حتى الثوابت الحرفية (التي تكتبها داخل علامات التنصيص المزدوجة "و") يتعامل معها Visual Basic .NET على انها كائنات من النوع String. جرب كتابة نقطة بعد أي ثابت حرفي لترى قائمة IntelliSense تظهر لك خصائص وطرق الفئة String **(شكل 6-1 في الصفحة التالية)**.



شكل 6-1: ظهور قائمة IntelliSense مع الثوابت الحرفية من النوع String.

لا تنسى ان الثوابت الحرفية (والتي هي كائنات) تحفظ في القسم Managed Heap من الذاكرة.

دعني اضيف شيء اخر مهم حول الثوابت الحرفية، مترجم اللغة Visual Basic .NET لا ينشئ إلا نسخة واحدة من كل ثابت حرفي، مما يعني انه حتى لو وجد اكثر من ثابت حرفي يحمل نفس القيمة، فلن يتم الاحتفاظ إلا بثابت واحد فقط، ففي السطر التالي:

```
Dim X As String = "ثـابـت حرفي"
Dim Y As String = "ثـابـت حرفي"

ArabicConsole.WriteLine(X Is Y) ' True
```

نكتشف ان المؤشران X و Y يشيران إلى كائن حرفي واحد فقط بالرغم من اننا عرفنا ثابتين حرفيين مختلفين. ليس هذا فقط، بل حتى عند استخدام معامل الدمج & فسينتج عنه نفس قيمة الثابت الحرفي اثناء الترجمة Compiling:

```
Dim X As String = "ثـابـت حرفي"
Dim Y As String = "ثـابـت" & " " & "حرفي"

ArabicConsole.WriteLine(X Is Y) ' True
```

PDF Factory Pro trial version www.fastio.com

الخصائص والطرق

لا تحتوي الفئات من النوع String إلا على خاصتين فقط هما Length و Chars، الخاصية الأولى تعود بعدد الحروف الموجودة في المتغير الحرفي، بينما الخاصية Chars تعود بقيمة من النوع Char تمثل الحرف الذي تحدد رقمه في السلسلة الحرفية (يبدأ الترقيم عادة من الصفر):

```
Dim X As String = "ABCDE"

ArabicConsole.WriteLine(X.Chars(0))                 ' A
ArabicConsole.WriteLine(X.Chars(X.Length - 1))      ' E
```

لا تنسى ان الحروف العربية تحفظ بترتيب معاكس للترتيب الذي تظهر به داخل محرر الشيفرة:

```
Dim x As String = "ا-ب-ت-ث"
Dim y As String = "A-B-C-D"


ArabicConsole.WriteLine(x.Chars(0)) ' ا
ArabicConsole.WriteLine(x.Chars(6)) ' ث

ArabicConsole.WriteLine(y.Chars(0)) ' A
ArabicConsole.WriteLine(y.Chars(6)) ' D
```

تحتوي الفئات من النوع Strings على العديد من الطرق Methods المخصصة للتعامل المباشر مع الحروف، تجد شرحا مفصلا في مستندات NET. Documentation، اما هنا فلن أقدم لك إلا عرض سريع ومختصر لبعض هذه الطرق. وقبل ان اعرض لك هذه الطرق، عليك معرفة ان قيمة المتغير الحرفي **لا تتأثر** أبدا، حيث ان استخدام هذه الطرق تؤدي إلى العودة **بقيمة حرفية جديدة** ولا تؤثر بأي حال على القيمة الأصلية التي يحملها المتغير الحرفي.

سأبدأ بالطريقة Trim() التي تمكنك من حذف جميع المسافات التي قد تكون موجودة في بداية ونهاية المتغير الحرفي:

```
Dim MyString As String = "  حذف المسافات  "

ArabicConsole.WriteLine(MyString.Trim) ' حذف المسافات
```

يمكنك إرسال وسيطة أخرى للطريقة Trim() من النوع Char (وهي غير محدودة العدد ParamArray) بحيث تحدد فيها الحروف التي تود حذفها في بداية ونهاية المتغير الحرفي:

PDF

```
Dim MyString As String = " ** حذف المسافات وعلامات النجمة ** "
ArabicConsole.WriteLine(MyString.Trim(" "c, "*"c))
```

بالإضافة إلى الطريقة Trim السابقة، توجد الطريقتين TrimStart() و TrimEnd() وهي لحذف المسافات قبل وبعد المتغير الحرفي.

انظر أيضا

ناقشت الوسيطات غير محدودة العدد ParamArray في الفصل الثاني **لغة البرمجة**.

الطريقتان StartsWith() و EndsWith() تمكنك من اختبار الحروف التي يبدأ او ينتهي بها المتغير الحرفي:

```
Dim MyName As String = "تركي العسيري"

' تحقق من الاسم الأول
If MyName.StartsWith("تركي") Then

    ' تحقق من الاسم الاخير
    If MyName.EndsWith("العسيري") Then
        …
        …
    End If

End If
```

تقوم الطريقة Insert() بإضافة (حشر) قيمة حرفية إلى قيمة المتغير الحرفي في الموقع الذي تحدده، إما الطريقة Remove() فهي تحذف عدداً معينا من الحروف من السلسلة الحرفية بالطول والموقع الذي تحدده عبر وسيطات هذه الطريقة:

```
Dim MyName As String = "تركي العسيري"

ArabicConsole.WriteLine(MyName.Insert(4, " ابراهيم "))' تركي ابراهيم العسيري
ArabicConsole.WriteLine(MyName.Remove(4, 8))  ' تركي
```

PDF

ملاحظة

ترقيم مواقع الحروف في الكائن الحرفي يبدأ بصفر.

وظيفة الطريقة Replace() هو استبدال قيمة حرفية بأخرى في المتغير الحرفي، فالاستدعاء التالي سيستبدل جميع الكلمات **محمد** بـ **محمد صلى الله عليه وسلم**:

```
Dim Story As String

…
…
ArabicConsole.WriteLine (Story.Replace ("محمد", "محمد صلى الله عليه وسلم"))
```

الطريقة Split() تقوم بتقسيم الجملة الحرفية إلى مصفوفة ويتم توزيعها استنادا إلى نوع القيمة التي تحددها كوسيطة ترسلها لهذه الطريقة، ففي الشيفرة التالية سيتم فصل كلمات الجملة وذلك لأني أرسلت الحرف **c" "** مع الوسيطة:

```
Dim MyString As String = "فصل كلمات هذه الجملة"

Dim Words As String() = MyString.Split(" "c)
Dim counter As Integer

For counter = 0 To UBound(Words)
    ArabicConsole.WriteLine(Words(counter))
Next
```

مخرجات الشيفرة السابقة ستكون:

```
فصل
كلمات
هذه
الجملة
```

أيضا، لديك الطريقة ToCharArray() والتي ستعيد لك مصفوفة من نوع Char تحوي السلسلة النصية.

```
Dim Words As Char() = MyString.ToCharArray()
```

PDF

اخيرا، الطريقة ()ToUpper موجه بشكل خاص للحروف الأبجدية الإنجليزية، حيث تقلب الحروف إلى الحروف الكبيرة Capital Letters بينما ()ToLower إلى الصغيرة:

```
Dim MyString As String = "I like Visual Basic .NET"

ArabicConsole.WriteLine(MyString.ToUpper) ' I LIKE VISUAL BASIC .NET
ArabicConsole.WriteLine(MyString.ToLower) ' I like visual basic .net
```

مقارنة الحروف

يمكنك إجراء المقارنات بين القيم الحرفية باستخدام الطريقة المشتركة ()Compare، حيث تعود بالقيمة 0 عند تساوي الوسيطتين المرسلتين، والقيمة 1 ان كانت الأولى اكبر من الثانية، والقيمة -1 ان كانت الأولى اصغر من الثانية:

```
Dim s1 As String = "AAA"
Dim s2 As String = "BBB"

Select Case String.Compare(s1, s2)
    Case 0
        ArabicConsole.WriteLine("s1 = s2")
    Case 1
        ArabicConsole.WriteLine("s1 > s2")
    Case -1
        ArabicConsole.WriteLine("s1 < s2")
End Select
```

أود ان انوه هنا بان المقارنة تشمل حالة الحروف الابجدية الانجليزيـــة Case-sensitive، حيث ان "turki" لا تساوي "TURKI" عند التعامـــل مـــع الطريقـــة ()Compare، وان أردت تجاهل حالة الحروف، يمكن إرسال الوسيطة True:

```
ArabicConsole.WriteLine(String.Compare("TURKI", "turki", True)) ' 0
```

وعند الحديث عن الحروف العربية، فيتوجب عليك معرفة ان علامات التشكيل لها اثر كبير عند المقارنة (لاحقا ستتعلم كيف يمكنك تجاهلها) بينما علامة الكشيدة سيتم تجاهلها حيث أن "محمد" تساوي "مـــحـــمـــد":

```
ArabicConsole.WriteLine (String.Compare ("كَلِمةٌ", "كلمة"))    ' 1
ArabicConsole.WriteLine(String.Compare("محمد" ,"مـــحـــمـــد"))  ' 0
```

PDF

ملاحظة

رغم ان الكشيدة تم تجاهلها في الشيفرة السابقة، إلا أن ذلك قد لا يحدث دائماً، فعند تجربتي للشيفرة السابقة كانت الاعدادات الإقليمية في نظام التشغيل لجهازي الشخصي هي **العربية** (**المملكة العربية السعودية**)، لذلك قد تظهر لك نتائج مخالفة عند تنفيذ الشيفرة على أنظمة مهيأة لمناطق أخرى.

توجد طريقة أخرى أسرع بخمس مرات من الطريقة ()Compare السابقة وهي الطريقة ()CompareOrdinal، حيث ان المقارنة هنا تتم استنادا إلى قيمة الحرف في جدول الترميز UNICODE وتتجاهل مقاييس ومواصفات اللغة الحالية (سواء العربية، الإنجليزية ...الخ):

```
ArabicConsole.WriteLine(String.CompareOrdinal ("محمد", "محمد")     ' 0
ArabicConsole.WriteLine(String.CompareOrdinal ("محمد", "مـحـمـد") ' -19
```

الفئة CultureInfo

في الفصل الأول **تعرف على Visual Basic .NET** ذكرت لك ان إطار عمل .NET Framework موجه إلى جميع لغات العالم الطبيعية وبما أن اللغة العربية إحدى هذه اللغات، فان نظام الفرز والمواصفات والمقارنة وتوزيع محارف لوحات المفاتيح العربية مأخوذة في الاعتبار، كذلك الحال مع مواصفات البيئة كتنسيق العملة، الوقت والتاريخ، الارقام، ...الخ.

عندما تريد ان تخاطب لغات العالم الطبيعية الأخرى في برامجك، فستجد كل ما تريده في مجال الاسماء System.Globalization، حيث يوفر لك هذا المجال مجموعة من الفئات التي يمكن أن استخدامها لإعطاء نكهة محلية لبرامجك ومشاريعك التي تنجزها في إطار عمل .NET Framework.

ستجد شرحا مفصلا لهذه الفئات في مستندات .NET Documentation، وبما انني لن أتجرأ على ترجمة مراجع Microsoft في هذا الكتاب، فستجد في ثنايا هذا الكتاب بعض الشروح التي تتناول بعض -وليس كل- هذه الفئات، والبداية ستكون مع الفئة CultureInfo.

تمكنك الفئة CultureInfo من محاكاة البيئة المحلية في الدولة، فإذا اردت تطبيق انظمة الدولة المعينة في برامجك (كتنسيق نظام العملة، اللغة الرسمية، الوقت والتاريخ ...الخ) عندما تنشئ كائن جديد من هذه الفئة، ارسل الاسم المختصر أو **رقم المعرف LCID** للغة والدولة مع

PDF Factory Pro trial version www.fastio.com

المشيد، وان كنت لا تعلم الاسم او رقم المعرف المناسب، يمكنك استدعاء الطريقة CurrentCulture() التي تعود بكائن يمثل الاعدادات الإقليمية في النظام الحالي. الشيفرة التالية ستزودنا بالاعدادات الإقليمية للجهاز الحالي، والمخرجات المعروضة أمامك تمثل الاعدادات الإقليمية لجهازي الشخصي:

```
' بافتراض انه تم استيراد مجال الاسماء التالي
Imports System.Globalization
…
…
' العودة بالاعدادات الاقليمية للجهاز الحالي
Dim KSA As CultureInfo = CultureInfo.CurrentCulture()

With KSA
    ' ar-SA
    ArabicConsole.WriteLine(.Name)
    ' Arabic (Saudi Arabia)
    ArabicConsole.WriteLine(.EnglishName)
    ' العربية (المملكة العربية السعودية)
    ArabicConsole.WriteLine(.NativeName)
    ' 1025
    ArabicConsole.WriteLine(.LCID)
End With
```

من الشيفرة السابقة، علمت ان **المملكة العربية السعودية** يرمز لها بالرمز **ar-SA** او رقم المعرف **1025**، لذلك أستطيع إنشاء كائن يحمل اعدادات المملكة العربية السعودية على الفور:

```
Dim KSA As New CultureInfo("ar-SA")
' او
Dim KSA As New CultureInfo(1025)
```

يمكنك استخدام الأسماء الأخرى لباقي الدول العربية والموضحة في الجدول بأعلى الصفحة التالية:

الرمز	المعرف	الدولة
ar-SA	0x0401	المملكة العربية السعودية
ar-DZ	0x1401	الجزائر.
ar-BH	0x3C01	البحرين.
ar-EG	0x0C01	مصر.
ar-IQ	0x0801	العراق.
ar-JO	0x2C01	الاردن.
ar-KW	0x3401	الكويت.
ar-LB	0x3001	لبنان.
ar-LY	0x1001	ليبيا.
ar-MA	0x1801	المغرب.
ar-OM	0x2001	عمان.
ar-QA	0x4001	قطر.
ar-SY	0x2801	سورية.
ar-TN	0x1C01	تونس.
ar-AE	0x3801	الامارات.
ar-YE	0x2401	اليمن.
ar	0x0001	باقي الدول العربية.

والان سأريك كيف يمكن الاستفادة من هذا الكائن لإجراء المقارنات، لنبدأ أولا بمعرفة مدى تأثيرها على الكشيدة، حيث أني في الملاحظة السابقة (صفحة 243) ذكرت ان تجاهل الكشيدة قد يختلف من نظام إلى آخر، ولكن الشيفرة التالية ستتجاهل الكشيدة عند المقارنة مهما اختلفت الاعدادات الإقليمية لنظام التشغيل الحالي:

```
' 0
ArabicConsole.WriteLine(String.Compare("محمد" ,"مـــحـــمـــد"), _
    , False, New CultureInfo("ar-SA")))
```

توجد خاصية في الفئة CultureInfo هي CompareInfo تحتوي على نسخة محسنة من الطريقة Compare() (الموجودة في الفئات من النوع String) بحيث تمكنك من تجاهل علامات التشكيل وذلك بارسال القيمة CompareOptions.IgnoreSymbols لها:

```
Dim KSA As New CultureInfo("ar-SA")

' 1
ArabicConsole.WriteLine(String.Compare("كَلِمةٌ" ,"كلمة")

' 0
ArabicConsole.WriteLine(KSA.CompareInfo.Compare("كَلِمةٌ" ,"كلمة", _
                        CompareOptions.IgnoreSymbols))
```

المزيد أيضا، يمكنك إرسال القيمة CompareOptions.IgnoreNonSpace ان رغبت في تجاهل الهمزة الموجودة على حرف الألف:

```
' 0
ArabicConsole.WriteLine(KSA.CompareInfo.Compare("أحمد" ,"احمد", _
    CompareOptions.IgnoreNonSpace))
```

لا تنسى ان القيم التي ترسلها في الوسيطة الثالثة ما هي إلا قيم لتركيبات من النوع Enum، لذلك تستطيع تجاهل علامات التشكيل وهمزة الألف في مقارنة واحدة ان استخدمت المعامل Or:

```
ArabicConsole.WriteLine(KSA.CompareInfo.Compare(s1, s2, _
    CompareOptions.IgnoreNonSpace Or CompareOptions.IgnoreSymbols))
```

ملاحظة

من احد عيوب إرسال القيمة CompareOptions.IgnoreSymbols إلى الطريقة CompareInfo.Compare()، هو ان عملية التجاهل ستشمل تجاهل علامات ورموز أخرى كالمسافات، علامات التنصيص، والرموز الخاصة مثل: رمز النسبة المئوية % ....الخ.

البحث عن الحروف

استخدامك للطريقة IndexOf() تمكنك من البحث عن كلمة في المتغير الحرفي، حيث تعود برقم يمثل موقع الحرف الأول من القيمة المرسلة، او القيمة -1 ان لم توجد القيمة التي تبحث عنها:

```
Dim x As String = "Can you find the word find?"

ArabicConsole.WriteLine(x.IndexOf("find"))   ' 8
ArabicConsole.WriteLine(x.IndexOf("Find"))   ' -1
```

الطريقة IndexOf() مرنة جدا فهي تسمح لك بتحديد نقطة البداية للبحث، لتتمكن من البحث عن مواقع جميع العبارات المتكررة:

```
Dim x As String = "Can you find the word find?"
Dim index As Integer = 0

Do
    index = x.IndexOf("find", index + 1)

    If index < 0 Then
        Exit Do
    Else
        ArabicConsole.WriteLine(index)
    End If
Loop
```

وعند الحديث عن الحروف العربية، فالكشيدة سيتم تجاهلها بحسب الاعدادات الإقليمية للنظام الحالي، بينما سيتم أخذ علامات التشكيل بعين الاعتبار:

```
Dim x As String = "هل تستطيع البحث عن هذه الكَلِمة او تلك الكلمة؟"

ArabicConsole.WriteLine(x.IndexOf("الكلمة")) ' 39
ArabicConsole.WriteLine(x.IndexOf("الكَلِمة")) ' 23
```

ولتجاهل علامات التشكيل، عد إلى الفئة CultureInfo مرة أخرى واستخدم نفس اسم الطريقة IndexOf() ولكن التابعة للخاصية CompareInfo مع إرسال القيمة CompareOptions.IgnoreSymbols:

```
Dim x As String = "هل تستطيع البحث عن هذه الكَلِمة او تلك الكلمة؟"
Dim KSA As New CultureInfo("ar-SA")

' 23
ArabicConsole.WriteLine(KSA.CompareInfo.IndexOf(x, "الكلمة", _
    CompareOptions.IgnoreSymbols)) ' 23
```

الفئات من النوع Char

لا يوجد الكثير لأخبرك به عن الفئة Char، ولكن عليك معرفة ان القيمة التي تحملها المتغيرات من النوع Char لا تتعدى حرف واحد، كما أن عليك إرفاق الذيل c مع الثابت عند إسناد هذا الحرف لتمييزه عن الثابت الحرفي من النوع String:

```
Dim X As Char = "A"c
Dim Y As Char = "B"    ' خطأ
```

تحتوي الفئات من النوع Char على مجموعة من الطرق المشتركة Shared Methods التي تمكنك من تحديد نوع الحرف (بالصيغة Isxxx())، وتعود بقيمة منطقية من النوع Boolean تمثل نتيجة الاختبار، اذكر لك بعضا منها:

```
' هل الحرف المرسل رقم
ArabicConsole.WriteLine(Char.IsDigit("1"c))          ' True
' هل الحرف المرسل حرف ابجدي
ArabicConsole.WriteLine(Char.IsLetter("ت"c))         ' True
' هل الحرف المرسل رقم او حرف
ArabicConsole.WriteLine(Char.IsLetterOrDigit("X"c))  ' True
' هل الحرف المرسل حرف ابجدي صغير
ArabicConsole.WriteLine(Char.IsLower("a"c))          ' True
' هل الحرف المرسل علامة تنصيص مفردة او مزدوجة
ArabicConsole.WriteLine(Char.IsPunctuation("""c))    ' True
```

يمكنك أيضا إرسال قيمة من النوع String إلى هذه الطرق ولكن لا تنسى إرسال وسيطة إضافية تمثل رقم الحرف المطلوب اختباره في الوسيطة الأولى المرسلة:

```
ArabicConsole.WriteLine(Char.IsDigit("A1", 0)) ' False
ArabicConsole.WriteLine(Char.IsDigit("A1", 1)) ' True
```

نقطة هامة أخيرة: البيانات من النوع Char هي بيانات من النوع ذو القيمة Value Type وليست مرجعية Reference Type كالبيانات من النوع String. لذلك، ضع الفروق بين البيانات المرجعية وذات القيمة في اعتبارك دائما عند التعامل مع هذه البيانات.

الفئات من النوع StringBuilder

كما ذكرت لك سابقا، البيانات الحرفية –سواء كانت ثوابت او متغيرات- لا تتغير قيمتها أبدا، وإنما يتم إنشاء نسخ جديدة من القيمة الحرفية ان دعت الحاجة (كاستدعاء طريقة تغير عدد او محتوى الحروف)، فالشيفرة التالية:

```
Dim x As String = "برعي ابو"

ArabicConsole.WriteLine(x)                          ' برعي ابو
ArabicConsole.WriteLine(x.Insert(8, " جبهة")) ' برعي ابو جبهة
ArabicConsole.WriteLine(x.Remove(4, 4))             ' برعي
```

PDF Factory Pro trial version www.fastio.com

قامت بإنشاء ثلاثة كائنات حرفية في ذاكرة Managed Heap هي "برعـــي ابـــو" و "جبهـــة" و "برعي ابو جبهة"، وكأنك قمت بإنشاء ثلاثة مؤشرات لها:

```
Dim x As String = "برعي ابو"
Dim y As String = x.Insert(8, " جبهة")
Dim z As String = x.Remove(4, 4)

ArabicConsole.WriteLine(x)     ' برعي ابو
ArabicConsole.WriteLine(y)     ' برعي ابو جبهة
ArabicConsole.WriteLine(z)     ' برعي
```

و بما ان القيم الحرفية هي من النوع المرجعي Reference Type فمن المؤكد أنه لن يتم تحرير ذاكرة Managed Heap منها إلا باستخدام المجموعة Garbage Collection. وفي كل مرة تسند قيمة او تعدل في محتوى قيمة متغير حرفي، سيتم إنشاء نسخة جديدة من الكـــائن ممـــا يؤدي إلى بطء في التعامل مع البيانات الحرفية بصفة عامة والحروف الطويلة بصفة خاصة.

انظر أيضا

كـان لـي حـديث مطـول عـن المجموعـة Garbage Collection وكيفيـة تحرير الذاكرة من البيانات المرجعية Reference Type سابقا في الفصل الثالث **الفئات والكائنات**.

من هنا يأتي دور الفئة System.Text.StringBuilder، حيث تمكنك هـــذه الفئـــة مـــن احتجاز مساحة ثابتة للقيم الحرفية (تسمى **String Buffer**) لتتم عملية تعـــديل قـــيم ومحتـــوى البيانات الحرفية في نفس المكان دون الحاجة إلى إنشاء نسخة جديدة من الكائن الحرفي.

يمكنك تحديد عدد الحروف في الـــ String Buffer عن طريق الخاصـــية Capacity او لحظة إنشاء الكائن وإرسال القيمة إلى وسيطة المشيد:

```
' حجم الـــ String Buffer 100 بـايت
' فـهو يحمل 50 حرف
Dim x As New System.Text.StringBuilder()
x.Capacity = 100

' او يمكنك إرسال عدد الحروف مع المشيد
Dim x As New System.Text.StringBuilder(50)
```

PDF

تستطيع استخدام معظم الطرق والخصائص الموجودة في الفئات الحرفية من النوع String للتعامل مع هذا المتغير (كـ Insert()، Remove() وغيرها) مع الإشارة إلى ان قيمة المتغير من النوع StringBuilder -خلافاً للبيانات الحرفية الأخرى- ستتأثر باستخدام هذه الطرق:

```
x.Insert(0, "برعي")
x.Insert(4, " ابو جبهة")

' تأثرت قيمة الكائن
ArabicConsole.WriteLine(x) ' برعي ابو جبهة
```

توجد طريقة إضافية في هذا النوع من الفئات هي Append() تمكنك من إضافة نصوص في نهاية السلسلة الحرفية:

```
Dim x As New System.Text.StringBuilder()
Dim counter As Integer

For counter = 1 To 9
    x.Append(counter)
Next

ArabicConsole.WriteLine(x) ' 123456789
```

من الضروري معرفة ان المتغيرات من النوع StringBuilder ليست كالمتغيرات الحرفية من النوع String، بل هي نوع خاص يختلف في بنيته التحتية عن الفئات من النوع String، فلا تحاول مثلا إسناد قيمة ثابت حرفي إلى متغير من النوع StringBuilder:

```
Dim Y As New System.Text.StringBuilder(50)

Y = "عباس السريع" ' رسالة خطأ
```

الفئات العددية

الفئات العددية هي مجموعة من الأنواع المعرفة في إطار عمل .NET Framework. تختلف في نوع وحجم القيمة العددية التي تسندها اليها وهي: Byte، Short، Integer، Long، Single، Double، و Decimal هذا النوع من البيانات من الأنواع ذات القيمة Value Type.

للأنواع Short، Integer، و Long مسميات أخرى تدل على حجمها هي: Int16، Int32، و Int64 على التوالي، قد تستخدمها لتضع حجم المتغيرات دائما في الاعتبار لحظة استخدامها.

PDF Factory Pro trial version www.fastio.com

انظر أيضا

تجد تفاصيل مجال القيم التي تسندها إلى المتغيرات العددية في الفصل الثاني **لغة البرمجة** وبالتحديد في الجدول الموجود بالصفحات 49-50.

الخصائص والطرق

جميع الفئات العددية تتشارك في خصائص وطرق مرتبطة بالأعداد والقيم التي تحملها المتغيرات العددية. وسأبدأ معك بالخاصيتين MaxValue و MinValue واللتان تعودان بقيمة تمثل اكبر و اصغر قيمة ضمن مجال الأعداد الذي يمكن للمتغير العددي ان يحمله، مع العلم ان هذه الخصائص للقراءة فقط ReadOnly ومشتركة Shared:

```
Dim X As Byte
Dim Y As Integer
Dim Z As Long
Dim W As Double

ArabicConsole.WriteLine(X.MinValue) ' 0
ArabicConsole.WriteLine(Y.MinValue) ' -2147483648
ArabicConsole.WriteLine(Z.MaxValue) ' 9223372036854775807
ArabicConsole.WriteLine(W.MaxValue) ' 1.79769313486232E+308
```

تحتوي الأنواع ذات الفاصلة العائمة Floating-point (كـ Single و Double) على خصائص إضافية كالخاصية Epsilon() والتي تعود بقيمة اصغر عدد عشري موجب –غير الصفر - يمكن للمتغير ان يحمله:

```
Dim X As Double
Dim Y As Single

ArabicConsole.WriteLine(X.MinValue) ' -1.79769313486232E+308
ArabicConsole.WriteLine(X.Epsilon)  ' 4.94065645841247E-324

ArabicConsole.WriteLine(Y.MinValue) ' -3.402823E+38
ArabicConsole.WriteLine(Y.Epsilon)  ' 1.401298E-45
```

PDF

تنسيق الأعداد

استدعائك للطريقة ()ToString يعود بالقيمة التي يحملها المتغير العددي ولكن من النوع الحرفي String:

```
Dim x As Integer = 100

ArabicConsole.WriteLine(x.ToString) ' 100
```

يمكنك إضافة مجموعة من الرموز لتنسيق الأعداد وإرسالها كوسيطة مع الطريقة ()ToString، كالرمز # الذي يمثل عدداً او مسافة خالية، او رمز الصفر **0** الذي يمثل عدداً او الرقم صفر:

```
Dim x As Integer = 10

ArabicConsole.WriteLine(x.ToString("####")) ' 10
ArabicConsole.WriteLine(x.ToString("0000")) ' 0010
```

توجد مجموعة إضافية من الرموز التي يمكنك استخدامها، تجد شرحا وافيا لها في مستندات NET Documentation. كالرمز "." الذي يمثل الفاصلة العشرية، او الرمز "," الذي يمثل فاصلة الآلاف، او الرمز "%" الذي يؤدي إلى عرض النسبة المئوية، او الرمز "**E**" الذي يمكنك من عرض الرقم في الصورة الأسية Exponential Form:

```
Dim X As Single = 100.1234
Dim Y As Integer = 9999999
Dim Z As Double = 0.1
Dim w As Long = 1500000000000

ArabicConsole.WriteLine(X.ToString("0000.#"))        ' 0100.1
ArabicConsole.WriteLine(Y.ToString("###,###,###"))   ' 9,999,999
ArabicConsole.WriteLine(Z.ToString("##.0 %"))        ' 10.0 %
ArabicConsole.WriteLine(w.ToString("### E+0"))       ' 150 E+10
```

من ناحية أخرى، توجد مجموعة جاهزة من التنسيقات العددية تتأثر بالاعدادات الإقليمية الحالية في الجهاز، المخرجات التالية مرتبطة بالاعدادات الإقليمية لجهازي الشخصي وهي **العربية (المملكة العربية السعودية)**:

```
Dim MyDouble As Double = 123456789

' Currency عملة
ArabicConsole.WriteLine(MyDouble.ToString("C")) ' 123,456,789.00 ر.س.
' Scientific علمي
ArabicConsole.WriteLine(MyDouble.ToString("E")) ' 1.234568E+008
```

```
' Percent النسبة المئوية
ArabicConsole.WriteLine(MyDouble.ToString("P")) ' 12,345,678,900.00%
' Number عدد
ArabicConsole.WriteLine(MyDouble.ToString("N")) ' 123,456,789.00
' Fixed-point فاصلة عائمة
ArabicConsole.WriteLine(MyDouble.ToString("F")) ' 123456789.00
```

وحتى تستخدم اعدادات إقليمية أخرى، أرسل الخاصية NumberFormat التابعة للفئة CultureInfo كوسيطة ثانية للطريقة ()ToString، المخرجات التالية من البيئة الالمانية (لاحظ الفرق في استخدام الفواصل والنقط):

```
Imports System.Globalization

…
…

Dim Germany As New CultureInfo("de-DE")
Dim MyDouble As Double = 123456789

With Germany
    ' 123.456.789,00 €
    ArabicConsole.WriteLine(MyDouble.ToString("C", .NumberFormat))
    ' 1,234568E+008
    ArabicConsole.WriteLine(MyDouble.ToString("E", .NumberFormat))
    ' 12,345,678,900.00%
    ArabicConsole.WriteLine(MyDouble.ToString("P", .NumberFormat))
    ' 123.456.789,00
    ArabicConsole.WriteLine(MyDouble.ToString("N", .NumberFormat))
    ' 123456789,00
    ArabicConsole.WriteLine(MyDouble.ToString("F", .NumberFormat))
End With
```

المزيد أيضا، الخاصية NumberFormat ما هي إلا كائن منشأ من الفئة NumberFormatInfo خصائصها قابلة للقراءة والكتابة بحيث يمكنك تخصيص نظام تنسيق للاعداد خاص بك:

```
Imports System.Globalization
…
Dim Custom As New NumberFormatInfo()
Dim X As Double = -12345.6789

' علامة الفاصلة العشرية ستكون فاصلة منقوطة
Custom.NumberDecimalSeparator = ";"
' علامة السالب ستكون نجمة
Custom.NegativeSign = "*"
ArabicConsole.WriteLine(X.ToString("", Custom)) ' *12345;6789
```

الفئة Math

تحتوي الفئة Microsoft.VisualBasic.Math على مجموعة كبيرة من الطرق المشتركة لإجراء العمليات الحسابية والرياضية على البيانات العددية، لا تنسى استيراد مجال الأسماء Microsoft.VisualBasic في مشروعك حتى تتمكن من الوصول إلى هذه الفئة بكتابة اسمها فقط:

```
Imports Microsoft.VisualBasic
```

من الطرق التابعة للفئة Math الطريقة Abs() التي تعود بالقيمة المطلقة Absolute Value، والطريقة Sqrt() للجذر التربيعي، والطريقة Pow() للاس، وطريقة اللوغارثم Log() او اللوغارثم العشري Log10()، والطريقة Sign() التي تعود بالقيمة 1 ان كان العدد موجب او -1 ان كان سالب او 0 ان كان صفر:

```
ArabicConsole.WriteLine(Math.Abs(-10))                  ' 10
ArabicConsole.WriteLine(Math.Sqrt(9))           ' 3
ArabicConsole.WriteLine(Math.Pow(2, 3))                 ' 8
ArabicConsole.WriteLine(Math.Log(9, 3))                 ' 2
ArabicConsole.WriteLine(Math.Log10(100))                ' 2
ArabicConsole.WriteLine(Math.Sign(Double.MinValue))     ' -1
ArabicConsole.WriteLine(Math.Sign(Double.Epsilon))      ' 1
```

الطريقة Sqrt() السابقة تعود بالجذر التربيعي للعدد، وان أردت الحصول على الجذر النوني فلا مخرج لك إلا بتطوير الطريقة NthSqr() بنفسك:

```
Function NthSqr(ByVal num As Double, ByVal root As Double) As Double
    Return num ^ (1 / root)
End Function

…
…

' مثال لاستدعائها
ArabicConsole.WriteLine(NthSqr(8, 3)) ' 2
```

طرق أخرى تتعامل مع الاعداد كالطريقة IEEERemainder() التي تعود بباقي القسمة، والطريقة Ceiling() التي تحذف الفاصلة العشرية وتعود بقيمة صحيحة اكبر من او تساوي العدد المرسل، بينما الطريقة Floor() تعود بقيمة صحيحة اصغر من او تساوي العدد المرسل:

```
ArabicConsole.WriteLine(Math.IEEERemainder(9, 2))    ' 1
ArabicConsole.WriteLine(Math.Ceiling(1.2))           ' 2
ArabicConsole.WriteLine(Math.Ceiling(-1.2))          ' -1
ArabicConsole.WriteLine(Math.Floor(1.2))             ' 1
ArabicConsole.WriteLine(Math.Floor(-1.2))            ' -2
```

يمكنك التحكم اكثر في خانة الفاصلة العشرية باستخدام طريقة التقريب Round() حيث تحدد عدد خانات الفاصلة العشرية:

```
ArabicConsole.WriteLine(Math.Round(1.1256, 1)) ' 1.1
ArabicConsole.WriteLine(Math.Round(1.1256, 2)) ' 1.13
ArabicConsole.WriteLine(Math.Round(1.1256, 3)) ' 1.126
```

اخيرا، طرق الدوال المثلثية Sin()، Cos()، Tan() ... الخ تعود بالقيمة المناسبة استنادا إلى الزاوية المرسلة بالراديان. راجع مستندات NET. Documentation لمزيد من التفاصيل حول هذه الطرق والطرق الأخرى.

توليد الأعداد العشوائية Random Numbers

تحتوي الفئة System.Random على مجموعة من الطرق التي تقوم بتوليد أعداد عشوائية، فمثلا الطريقة Next() تعود بقيمة عشوائية صحيحة Integer موجبه:

```
Dim Rnd As New Random()
Dim counter As Integer

For counter = 1 To 10
    ArabicConsole.WriteLine(Rnd.Next) ' 2345684
Next
```

اما الطريقة NextDouble() فهي تعود بعدد عشوائي عشري Double مجاله من 0 إلى 1:

```
Dim Rnd As New Random()
Dim counter As Integer

For counter = 1 To 10
    ArabicConsole.WriteLine(Rnd.NextDouble) ' 0.01234865
Next
```

المزيد أيضا، يمكنك تخصيص مجال معين للاعداد العشوائية بارسال اصغر قيمة واكبر قيمة كوسيطات مع الطريقتين السابقتين:

```
Dim Rnd As New Random()
Dim counter As Integer

For counter = 1 To 10
    ' توليد اعداد عشوائية مجالها من -10 إلى 10 '
    ArabicConsole.WriteLine(Rnd.Next(-10, 10))
Next
```

اخيرا، يمكنك ملء مصفوفة عددية من النوع Byte بأرقام عشوائية في خطوة واحدة باستخدام الطريقة NextBytes():

```
Dim Rnd As New Random()
Dim counter As Integer
Dim x(99) As Byte

Rnd.NextBytes(x)

For counter = 0 To UBound(x)
    ArabicConsole.WriteLine(x(counter))
Next
```

فئات أخرى

سأتناول في هذا القسم بعض الفئات التي تطرقت لها سابقا في الفصل الثاني **لغة البرمجة**.

فئات الوقت والتاريخ

تعريفك لمتغير من النوع Date يعني تعريفك لمتغير من الفئة System.DateTime، وهما يمثلان نفس الفئة:

```
Dim today As Date
Dim yesterday As DateTime
```

المتغيران today و yesterday يحملان قيمة تشمل وقت Time وتاريخ Date، يمكنك إسناد هذه القيمة لحظة تعريف المتغير بعدة اساليب: كارسال وسيطات السنة، الشهر، واليوم على التوالي مع المشيد:

```
' 26 سبتمبر لعام 2002
' الساعة 12:00 صباحا
Dim today As New Date(2002, 9, 26)
```

الوقت الذي يحمله المتغير السابق سيكون الساعة 12:00 صباحا، ويمكن تحديد الوقت بإرسال الساعة، الدقيقة، والثانية على التوالي:

```
' 26 سيبتمبر لعام 2002
' الساعة 1:30 مساء
Dim today As New Date(2002, 9, 26, 13, 30, 0)
```

او يمكنك إسناد الخاصية المشتركة Now() التي تعود بالوقت والتاريخ الحالي:

```
' 26 سيبتمبر لعام 2002
' الساعة 3:30:23 صباحا
Dim today As Date = Date.Now
```

او إرسال ثابت الوقت والتاريخ بين علامتي # و #:

```
' 20 فبراير لعام 2003
' الساعة 2:00:00 صباحا
Dim today As Date = #1/20/2003 2:00:00 AM#
```

بعد إسنادك لقيمة ابتدائية لمتغير التاريخ، يمكنك الاستعلام عن أجزاء من القيمة عن طريق مجموعة من الخصائص (للقراءة فقط ReadOnly) هي Year، Month، Day، Hour، Minute، و Second تمثل السنة، الشهر، اليوم، الساعة، الدقيقة، والثانية على التوالي:

```
Dim today As New Date(2002, 9, 26, 13, 30, 0)

ArabicConsole.WriteLine(today.Year)      ' 2002
ArabicConsole.WriteLine(today.Month)   ' 9
ArabicConsole.WriteLine(today.Day)       ' 26
ArabicConsole.WriteLine(today.Hour)      ' 13
ArabicConsole.WriteLine(today.Minute)  ' 30
ArabicConsole.WriteLine(today.Second)  ' 0
```

وعند الحديث عن العمليات الرياضية على متغير من نوع Date، فتوجد مجموعة من الطرق لتضيف إلى قيمة المتغير الحالي سنة، شهر، يوم، ساعة، دقيقة، او ثانية وهي: AddYears()، AddMonths()، AddDays()، AddHours()، AddMinutes()، و AddSeconds():

PDF Factory Pro trial version www.fastio.com

```
Dim today As New Date(2002, 9, 26)
Dim adate As Date

' اضافة 10 ايام ليكون التاريخ
' 2002/10/06
adate = today.AddDays(10)
```

وان رغبت في إنقاص قيمة من التاريخ او الوقت، فاستخدم نفس الطرق السابقة ولكن مع إرسال أرقام سالبة:

```
Dim today As New Date(2002, 9, 26)
Dim adate As Date

' انقاص 27 يوم ليكون التاريخ
' 2002/8/30
adate = today.AddDays(-27)
```

أخيرا، توجد طريقة مشتركة قد تفيدك كثير هي DaysInMonths() تعود بعدد الأيام الموجودة في الشهر الحالي:

```
ArabicConsole.WriteLine(Date.DaysInMonth(2002, 1)) ' 31
```

دعم التقويم الهجري:

ان جميع العمليات التي قمنا بها في السطور السابقة (من بداية إسناد القيم للمتغير حتى الاستعلام عن القيمة التي يحملها)، كانت تعتمد –بشكل افتراضي- على التقويم الميلادي، ولكن ان حاولت طباعة قيمة المتغير، فان المخرجات ستتأثر بالاعدادات الإقليمية الحالية:

```
' ادخلت القيمة هنا بالتقويم الميلادي
Dim today As New Date(2002, 9, 26)

' ظهرت المخرجات استنادا إلى الاعدادات الاقليمية الحالية
' العربية (المملكة العربية السعودية)
' بالتقويم الهجري
ArabicConsole.WriteLine(today) ' 1423/7/20
```

السبب الذي قد يفسر لك عملية التحويل السابقة تقني، حيث ان القيمة التي يتعامل معها المتغير today السابق تستخدم التقويم الميلادي، ولكن عند الإخراج تم استدعاء الطريقة ToString() والتي تتأثر بالاعدادات الاقليمية الحالية في النظام.

يمكنك تحويل المتغير التاريخي إلى متغير يعتمد التقويم الهجري بارسال الكائن Calendar التابع للفئة CultureInfo:

```
Imports System.Globalization

…
…

Dim KSA As New CultureInfo("ar-SA")

' القيم المدخلة هنا تعتمد التقويم الهجري
Dim today As New Date(1423, 7, 20, KSA.Calendar)
```

لماذا تم اعتماد التقويم الهجري في المتغير today السابق –رغم ان اعدادات **العربية (المملكة العربية السعودية)** تحتوي على 6 تقاويم؟ والجواب هو لان التقويم الهجري هو التقويم **الافتراضي** في اعدادات **العربية (المملكة العربية السعودية)**، ولو حاولت تطبيق اعدادات **العربية (مصر)**، فعليك الحذر كل الحذر حيث ان التقويم الافتراضي لها هو الميلادي وليس الهجري:

```
Imports System.Globalization

…
…

Dim Egypt As New CultureInfo("ar-EG")

' تنبيه: القيم المدخلة هنا تعتمد التقويم الميلادي
' وليس الهجري
Dim today As New Date(1423, 7, 20, Egypt.Calendar)
```

الغرض من التنبيه السابق هو نصيحتك لاعتماد التقويم الهجري –ان كنت ترغب- بدون الاعتماد على الاعدادات الاقليمية للجهاز كالطريقة السابقة، وانما تعريف متغير لكائن من الفئة HijriCalendar بشكل مباشر:

```
Imports System.Globalization
…
…
Dim hijra As New HijriCalendar()

' القيم المدخلة هنا تعتمد التقويم الهجري
Dim today As New Date(1423, 7, 20, hijra)
```

الفئة HijriCalendar تمثل التقويم الهجري بكل ما تحمل الكلمة من معنى وهي مشتقة من الفئة Calendar، لذلك فهي تشتق جميع الطرق والخصائص. مع ذلك، توجد مجموعة كبيرة من الطرق تم إعادة قيادتها Overrides في الفئة HijriCalendar -وبعضها تم إعادة تعريفها Overloads أيضا- كـ AddMonths()، AddYears()، GetDayOfMonth()، GetDayOfWeek() ... الخ (راجع مستندات NET. Documentation. لمزيد من التفاصيل حول التعديلات التي طرأت عليها).

عدد الايام في الشهر الحالي يختلف من سنة إلى سنة ومن شهر إلى شهر، لذلك عليك استدعاء الطريقة GetDaysInMonth() لمعرفة عدد الايام في الشهر الحالي:

```
Dim hijra As New HijriCalendar()

ArabicConsole.WriteLine(hijra.GetDaysInMonth(1423, 7)) ' 30
ArabicConsole.WriteLine(hijra.GetDaysInMonth(1423, 8)) ' 29
```

وعند الحديث عن شهر رمضان المبارك، فبكل تأكيد لن تعتمد على إطار عمل .NET Framework في تحديد عدد ايام الشهر الحالي، حيث ان رسول الله صلى الله عليه وسلم امرنا في حديثه الشريف بالاعتماد على رؤية الهلال وليس على الحساب الفلكي الذي تعتمده أجهزة الحاسب (فقد يكون الشهر 29 او 30 يوماً). لذلك، عليك إعلام المستخدم بانه يتوجب عليه إجراء التعديلات اللازمة للتقويم الهجري من لوحة التحكم **(شكل 6-2 بالصفحة المقابلة)**، او أن تحصر المسؤولية عليك –كمبرمج عربي- في إجراء التعديلات اللازمة بالتوغل داخل مسجل النظام Windows Registry وتعديل قيمة المفتاح:

```
HKEY_CURRENT_USER\Control Panel\International\AddHijriDate
```

PDF Factory Pro trial version www.fastio.com

شكل 6-2: تعديل قيمة التاريخ الهجري من Regional Options في لوحة التحكم Control Panel.

انظر أيضا

الفصل السادس عشر **مواضيع متقدمة** يعرض لك مجموعة من الفئات تمكنك من تعديل مسجل النظام Windows Registry.

تنسيق الوقت والتاريخ:

باختصار شديد، يمكنك استخدام مجموعة من الطرق التي تعود بقيمة حرفية للوقت والتاريخ الذي يحمله المتغير:

```
Dim x As Date = Date.Now

ArabicConsole.WriteLine(x.ToShortDateString) ' 26/09/2002
ArabicConsole.WriteLine(x.ToLongDateString)  ' 26 September, 2002
ArabicConsole.WriteLine(x.ToLongTimeString)  ' 10:41:32 م
ArabicConsole.WriteLine(x.ToShortTimeString) ' 10:41 م
```

PDF

مخرجات الطرق السابقة تتأثر باعدادات النظام الحالي، فقد تظهر لك نتائج مختلفة في جهازك الخاص، إن أردت تنسيق طريقة عرض الوقت التاريخ بنفسك يمكنك استخدام الطريقة ToString():

```
Dim X As New Date(2002, 12, 20, 23, 30, 0)

' 2/12/20
ArabicConsole.WriteLine(X.ToString("y/M/d"))
' 02/12/20
ArabicConsole.WriteLine(X.ToString("yy/MM/dd"))
' 2002/Dec/Fri
ArabicConsole.WriteLine(X.ToString("yyy/MMM/ddd"))
' 2002/December/20 Friday
ArabicConsole.WriteLine(X.ToString("yyyy/MMMM/d dddd"))
' 11:30:00 P
ArabicConsole.WriteLine(X.ToString("hh:mm:ss t"))
' 11:30:00 PM
ArabicConsole.WriteLine(X.ToString("hh:mm:ss tt"))
' 23:30:00
ArabicConsole.WriteLine(X.ToString("HH:mm:ss"))
' 2002/30/20 11:30:00 PM
ArabicConsole.WriteLine(X.ToString("yyyy/m/d hh:mm:ss tt"))
```

ملاحظة

رمز الحروف الكبيرة MM يمثل شهر، بينما رمز الحروف الصغيرة mm فيمثل دقيقة.

من ناحية أخرى، اللغة المستخدمة (في أسماء الأشهر او الايام ...الخ) ونوع التقويم (هجري، ميلادي، وغيرها) تعتمد اعتماداً كلياً على اعدادات النظام الحالي. أما إن أردت التحكم فيها، فلا بد لك من استخدام الفئة DateTimeFormatInfo (بالمناسبة، الخاصية DateTimeFormat والتابعة للفئة CultureInfo كائن من النوع DateTimeFormatInfo):

```
Dim X As New Date(2002, 12, 20)
Dim arabicDateFormat As DateTimeFormatInfo

arabicDateFormat = New CultureInfo("ar-SA").DateTimeFormat
arabicDateFormat.Calendar = New HijriCalendar()

' 1423/شوال/16 الجمعة
ArabicConsole.WriteLine(X.ToString("yyyy/MMMM/d dddd", _
      arabicDateFormat))
```

PDF

انصحك بإلقاء نظرة إلى مستندات Documentation .NET. للحصول على المزيد من التفاصيل حول استخدام الفئة القوية DateTimeFormatInfo.

الفئات من النوع Enum

جميع التركيبات من النوع Enum التي تعرفها في شيفراتك المصدرية، هي مشتقة –بشكل تلقائي- من الفئة System.Enum، والطريقة الوحيدة التي يمكنك .NET Visual Basic من تعريف تركيب من النوع Enum هو باستخدام الكلمة المحجوزة Enum كما رأينا في الفصل الثاني **لغة البرمجة**:

```
Enum Programmer
    VisualBasic
    CSharp
    CPlusPlus
    Java
    Delphi
End Enum
```

الطريقة ()ToString التابعة للفئات من النوع Enum تم إعادة قيادتها Overrides بحيث تعود بقيمة تمثل اسم العضو الذي تم تعريفه في التركيب عوضا عن القيمة التي يحملها:

```
Dim Turki As Programmer

Turki = Programmer.VisualBasic

ArabicConsole.WriteLine(Turki)              ' 0
ArabicConsole.WriteLine(Turki.ToString)     ' VisualBasic
```

على عكس الطريقة ()ToString السابقة، توجد الطريقة المشتركة ()Parse التي تمكنك من إسناد قيمة للمتغير بكتابة اسم العضو حرفيا:

```
' في حالة
' Option Strict Off
Turki = [Enum].Parse(GetType(Programmer), "CSharp")

ArabicConsole.WriteLine(Turki)              ' 1
ArabicConsole.WriteLine(Turki.ToString)     ' CSharp
```

كما في التعليق السابق، لن تعمل الشيفرة السابقة إلا عند تعطيل العبارة Option Strict، اما ان كانت مفعلة فعليك العود إلى استخدام المعامل ()CType:

```
' في حالة
' Option Strict On
Turki = CType([Enum].Parse(GetType(Programmer), "CSharp"), _
        Programmer)

ArabicConsole.WriteLine(Turki)                ' 1
ArabicConsole.WriteLine(Turki.ToString)       ' CSharp
```

ضع في اعتبارك ان الطريقة Parse() تتأثر بحالة الأحرف الكبيرة والصغيرة Case-Sensitive، مع ذلك تستطيع تجاهل حالة الاحرف بإرسال القيمة True:

```
Turki = [Enum].Parse(GetType(Programmer), "CSharp", True)
```

القيم التي تحملها المتغيرات المعرفة من التركيبات من النوع Enum قيم عددية، بحيث يمكنك إسنادها مباشرة إلى المتغير:

```
Dim Turki As Programmer

Turki = CType(4, Programmer)

ArabicConsole.WriteLine(Turki)                ' 4
ArabicConsole.WriteLine(Turki.ToString)       ' Delphi
```

يفضل استخدام الطريقة IsDefined() للتحقق من دعم التركيب إلى العدد قبل إسناده للمتغير:

```
If [Enum].IsDefined(GetType(Programmer), 3) Then
    Turki = CType(3, Programmer)
End If
```

اخيرا، يمكنك الاستعلام عن جميع القيم واسماء الأعضاء التابعة للتركيب باستخدام الطريقتين GetNames() و GetValues(). الأولى تعود بمصفوفة من النوع String تحمل اسماء أعضاء التركيب والثانية بمصفوفة من النوع Object تمثل قيم تلك الأعضاء:

```
Dim names() As String = [Enum].GetNames(GetType(Programmer))
Dim values As Array = [Enum].GetValues(GetType(Programmer))
Dim counter As Integer

For counter = 0 To UBound(names)
    ArabicConsole.WriteLine(names(counter) & " = " & _
         CInt(values.GetValue(counter)))
Next
```

PDF Factory Pro trial version www.fastio.com

مخرجات الشيفرة الموجودة في أسفل الصفحة السابقة ستكون كالتالي:

```
VisualBasic = 0
CSharp = 1
CPlusPlus = 2
Java = 3
Delphi = 4
```

تركيبات Enum من النوع Bit-Coded:

في البرامج الجدية، يتم تعريف تركيبات Enum على انها تركيبات **بتية Bit-Coded** بحيث يمكن للمتغير الواحد الاحتفاظ باكثر من قيمة يتم دمجها باستخدام المعاملات المنطقية Or، And، XOr ...الخ. لعمل ذلك، أضف المواصفة Flags Attribute عند تعريف التركيب:

```
<Flags()> _
Enum Programmer
    None = 0
    VisualBasic = 1
    CSharp = 2
    CPlusPlus = 4
    Java = 8
    Delphi = 16
End Enum
```

معظم الطرق السابق ذكرها ستتأثر إن تم استخدام هذه المواصفة، فمثلا الطريقة ()ToString ستستخلص اسماء جميع الأعضاء التي يمثلها المتغير:

```
Dim Ali As Programmer

Ali = Programmer.VisualBasic Or Programmer.CSharp

ArabicConsole.WriteLine(Ali)            ' 3
ArabicConsole.WriteLine(Ali.ToString) ' VisualBasic, CSharp
```

الفئات من النوع Array

تحدثت بما فيه الكفاية في الفصل الثاني **لغة البرمجة** عن المصفوفات وطريقة التعامل معها. وسأضيف في هذه الفقرة أن جميع المصفوفات التي تعرفها في برنامجك، تصبح مشتقة وراثيا – بشكل تلقائي- من الفئة System.Array، ودعني أذكرك هنا بان جميع المصفوفات من النوع المرجعي Reference Type حتى وان كان نوع البيانات من ذات القيمة Value Type، الشيفرة التالية خير دليل:

```
Dim X() As Integer = {1, 2, 3}
Dim Y As Array = X

Y.SetValue(100, 0)
Y.SetValue(200, 1)
Y.SetValue(300, 2)


' X() لاحظ ان قيمة عناصر المصفوفة
' قد تغيرت
ArabicConsole.WriteLine(X(0)) ' 100
ArabicConsole.WriteLine(X(1)) ' 200
ArabicConsole.WriteLine(X(2)) ' 300
```

اما ان اردت نسخ قيم المصفوفات وليس مؤشراتها، فيمكنك استخدام الطريقة ()Clone التي تم ذكرها سابقا:

```
Dim X() As Integer = {1, 2, 3}
Dim Y() As Integer = CType(X.Clone, Integer())

Y(0) = 100
Y(0) = 200
Y(0) = 300

' لاحظ ان قيمة العناصر لم تتغير في
' X() المصفوفة
ArabicConsole.WriteLine(X(0)) ' 1
ArabicConsole.WriteLine(X(1)) ' 2
ArabicConsole.WriteLine(X(2)) ' 3
```

من خصائص الفئة Array الخاصية Rank التي تمكنك من الاستعلام عن عدد أبعاد المصفوفة، والخاصية Length تعود بقيمة تمثل عدد العناصر التي يمكن ان تحتويها المصفوفة، بينما الخاصية GetLenght تعود بقيمة تمثل عدد العناصر في البعد المعين التابع للمصفوفة:

```
Dim X(9, 5) As Integer

ArabicConsole.WriteLine(X.Rank)              ' 2
ArabicConsole.WriteLine(X.Length)            ' 60
ArabicConsole.WriteLine(X.GetLength(0))      ' 10
ArabicConsole.WriteLine(X.GetLength(1))      ' 6
```

المزيد أيضا، عند تطبيق حلقة For Each … Next على المصفوفات، كن متأكداً بان عملية المسح على عناصر المصفوفة ستتم بشكل أفقي وليس عمودي (أي أن البداية ستكون من

العنصر (0, 0) فالعنصـــر (0, 1) فالعنصر (0, x) فالعنـــصر (1, 0) فالعنـــصر (1, 1) .... وهكذا:

```
Dim records(,) As String = {{"تركي", "احمد", "خالد"}, _
                            {"الكويت", "الامارات", "السعودية"}}

Dim field As String

For Each field In records
    ArabicConsole.WriteLine(field)
Next
```

مخرجات الشيفرة السابقة ستكون كالتالي:

```
تركي
احمد
خالد
السعودية
الامارات
الكويت
```

فرز عناصر المصفوفة:

لن تجد في عالم .NET. أسرع وافضل من الطريقة ()Sort إن أردت فرز عناصر المصفوفة:

```
Dim X() As Integer = {5, 2, 9}
Dim counter As Integer

Array.Sort(X)

For counter = 0 To UBound(X)
    ArabicConsole.WriteLine(X(counter))
Next
```

الشيفرة السابقة ستقوم بترتيب عناصر المصفوفة بشكل تصاعدي (من الصغير إلى الكبير)، اما ان أردت جعل الفرز تنازلي فاستدعي الطريقة ()Reverse مباشرة بعد عملية الفرز:

```
Dim X() As Integer = {5, 2, 9}
Dim counter As Integer

Array.Sort(X)
Array.Reverse(X)
…
…
```

PDF Factory Pro trial version www.fastio.com

من الأشياء الظريفة في الطريقة Sort() هو إمكانية تحديد العناصر التي تود فرزها، فالاستدعاء التالي سيقوم بفرز العناصر الثلاثة الوسطى فقط لعدم الحاجة لفرز العناصر الأخرى – مما يزيد من سرعة التنفيذ يا عسل:

```
Dim X() As Integer = {1, 2, 3, 6, 4, 5, 7, 8, 9}
Dim counter As Integer

' فرز العناصر الثلاث الوسطى فقط
Array.Sort(X, 3, 5)

For counter = 0 To UBound(X)
    ArabicConsole.WriteLine(X(counter))
Next
```

إلى جانب كفاءة الطريقة Sort() في ترتيب العناصر العددية، فيسرني إخبارك بان هذه الطريقة تعمل بنفس الكفاءة مع العناصر الحرفية أيضا، حيث انها تقوم بترتيب العناصر استنادا إلى المقابل العددي للحروف في جدول UNICODE:

```
Dim X() As String = {"Turki", "Ali", "Basmah"}
Dim counter As Integer

Array.Sort(X)

For counter = 0 To UBound(X)
    ArabicConsole.WriteLine(X(counter))
Next
```

وعند الحديث عن لغتنا الجميلة، فيسرني أيضا اخبارك بان الطريقة Sort() تتجاهل علامة الكشيدة وعلامات التشكيل بحيث لا تخل بعملية الفرز، فلا تحمل هم فرز الحروف العربية.

ملاحظة

رغم ان عملية الفرز تتم استنادا إلى المقابل العددي للحرف الأبجدي، إلا ان الطابع اللغوي يغلب اكثر، حيث يتم تجاهل حالة الاحرف الصغيرة Small والكبيرة Capital للحروف الإنجليزية. لذلك، لا تعتقد ان حرف a الصغير قد يتبع حرف Z الكبير.

البحث في عناصر المصفوفة:

استخدم الطريقة IndexOf() لتعود برقم فهرس العنصر في المصفوفة، ستفيدك هذه الطريقة كثيرا ان رغبت في البحث عن عنصر من عناصر المصفوفة، وان لم تجد القيمة المرسلة في الوسيطة الثانية، ستعود بالقيمة -1 مع العلم ان الطريقة تتأثر بحالة الأحرف الكبيرة والصغيرة الإنجليزية:

```
Dim X() As String = {"Turki", "Ali", "Ahmed"}

ArabicConsole.WriteLine(Array.IndexOf(X, "Ahmed")) ' 2
ArabicConsole.WriteLine(Array.IndexOf(X, "turki")) ' -1
```

بالنسبة للحروف العربية، فالطريقة IndexOf() تتأثر –مع الأسف- بعلامات الكشيدة والتشكيل، لذلك ستعود بالقيمة -1 ان لم تكن القيم متطابقة تماما من ناحية علامات التشكيل والكشيدة:

```
Dim X() As String = { "تركي", "احمدُ", "خالد"}

ArabicConsole.WriteLine(Array.IndexOf(X, "تركي"))    ' -1
ArabicConsole.WriteLine(Array.IndexOf(X, "احمد"))    ' -1
ArabicConsole.WriteLine(Array.IndexOf(X, "خالد"))    ' -1
```

اخيرا، الطريقة IndexOf() تستخدم خوارزم البحث الخطي Linear Search والذي يعيبه البطء الشديد خاصة مع المصفوفات الكبيرة، لذلك أنصحك بشدة استخدام البحث الثنائي Binary Search خصوصا مع المصفوفات كبيرة الحجم عن طريق استدعاء الطريقة BinarySearch() **والتي لن تعود بقيمة صحيحة إلا إن كانت المصفوفة مرتبة:**

```
Dim X() As String = {"Turki", "Ali", "Ahmed"}

ArabicConsole.WriteLine(Array.BinarySearch(X, "Ahmed"))  ' -1
' لابد من ترتيب عناصر المصفوفة لتطبيق
' البحث الثنائي عليها
Array.Sort(X)
ArabicConsole.WriteLine(Array.BinarySearch(X, "Ahmed"))  ' 0
```

مجال أسماء System.Collections

يحتوي مجال الأسماء System.Collections على عشرات الفئات الخاصة باحتواء البيانات المختلفة تسمى **المجموعات Collections**. صحيح ان الفئات من النوع Array مرنة بما فيه الكفاية لاحتواء البيانات، إلا ان استخدام الفئات المشمولة في مجال الاسماء System.Collections يعتبر إجراء أكثر حرفية، لما توفره لك هذه الفئات من طرق وخصائص لن تستطيع تطبيق الخوارزميات البرمجية المعقدة دون الاعتماد عليها.

يمكنك تعلم كل فئة من هذه الفئات على حدة، ولكني اعتقد أن إتقان الواجهات Interfaces التي تشملها هذه الفئات سيسهل عليك فهمها، كما أنه يختصر الوقت.

الواجهات ICollection و IList

جميع الفئات في مجال الاسماء System.Collections تحتوي على الواجهة ICollection وهي واجهة مشتقة وراثياً من الواجهة IEnumerable (التي لمحت اليها في الفصل السابق) مما يعني إمكانية تطبيق الحلقة For Each على هذه الفئات.

تحتوي الواجهة ICollection على الخاصية Count التي تعود بعدد العناصر في المجموعة، كما تحتوي أيضا على الطريقة CopyTo() التي تمكنك من نسخ البيانات إلى مصفوفة Array.

توجد واجهتان مشتقتان وراثيا من الواجهة ICollection هما IList و IDictionary. سأذكر هنا الواجهة IList فقط والتي تحتوي على مجموعة من الطرق منها Add() لإضافة عناصر جديدة، Insert() لتعديل قيمة عنصر، Remove() لحذف عنصر، Clear() لحذف جميع العناصر وغيرها من الطرق.

ملاحظة

بعض الفئات تقوم بتضمين الواجهات السابقة ولكن لا تظهر كافة أعضائها، فمثلا المصفوفات (والمنشئة من الفئة Array) تحتوي على الواجهة IList ولكنها لا تظهر كل أعضاء الواجهة (كالطرق Add() و Remove()).

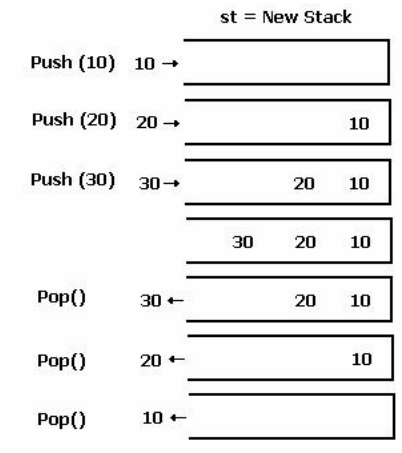
PDF Factory Pro trial version www.fineprint.com

سأقوم الآن بعرض سريع ومختصر لثلاث فئات موجودة في مجال اسماء System.Collections، انصحك بشدة بالإبحار في مكتبة MSDN للحصول على شرح لجميع الطرق والخصائص المدعومة بها والفئات الأخرى من نفس مجال الأسماء، وتذكر انه يمكن لهذه المجموعات ان تحتوي على جميع الأنواع المختلفة من البيانات. كما تستطيع أيضا تطوير فئات خاصة بك وذلك بتضمين الواجهات السابق ذكرها فيها.

الفئة Stack

تمكنك الفئة Stack من تطبيق خوارزم Last-In-First-Out (LIFO) حيث يكون أول عنصر يضاف إلى هذه المجموعة هو اخر عنصر يخرج منها. استخدم الطريقة Push() لإضافة العنصر والطريقة Pop() لقراءة العنصر:

```
Dim st As New Stack(100)

st.Push(10)
st.Push(20)
st.Push(30)

ArabicConsole.WriteLine(st.Pop) ' 30
ArabicConsole.WriteLine(st.Pop) ' 20
ArabicConsole.WriteLine(st.Pop) ' 10
```

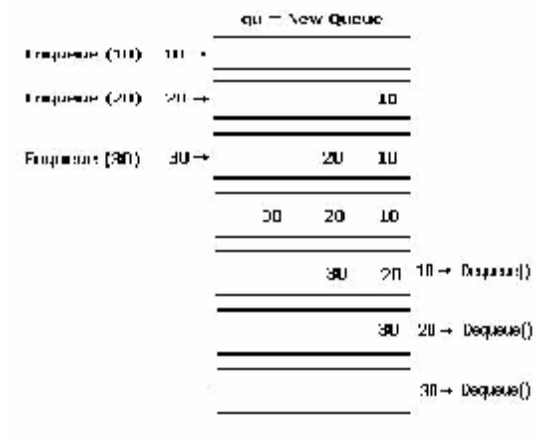


شكل 6-3: شكل توضيحي لبيانات الكائن st.

الفئة Queue

تمكنك الفئة Queue من تطبيق خوارزم First-In-First-Out (FIFO) حيث يكون أول عنصر يضاف إلى هذه المجموعة هو أول عنصر يخرج منها. استخدم الطريقة ()Enqueue لإضافة العنصر والطريقة () Dequeue لقراءة العنصر:

```
Dim qu As New Queue(100)

qu.Enqueue(10)
qu.Enqueue(20)
qu.Enqueue(30)

ArabicConsole.WriteLine(qu.Dequeue) ' 10
ArabicConsole.WriteLine(qu.Dequeue) ' 20
ArabicConsole.WriteLine(qu.Dequeue) ' 30
```



شكل 6-4: شكل توضيحي لبيانات الكائن qu.

الفئة ArrayList

تحتوي الفئات من النوع ArrayList على الواجهة IList، يمكن اعتبار الفئات من النوع ArrayList كالمصفوفات التقليدية (الفئات من النوع Array)، ولكنك سرعان ما ستكتشف بعض المزايا الإضافية في هذه الفئات بعد قراءتك للسطور التالية.

عند إنشاء كائن من هذه الفئة، يفضل تحديد العدد الابتدائي لعناصر المجموعة او سيكون 16 عنصر بشكل افتراضي مع العلم انك تستطيع التحكم في عدد العناصر في أي وقت بإسناد قيمة إلى الخاصية Capacity:

```
Dim X As New ArrayList()        ' 16 عنصر بشكل افتراضي
Dim Y As New ArrayList(100)     ' 101 عنصر وليس 100
```

يمكنك البدء في تعيين قيم العناصر باستخدام الطريقة Insert():

```
' X العنصر الأول للمجموعة
X.Insert (0, "تركي")
' Y العنصر الاخير للمجموعة
Y.Insert (99. "Visual Basic .NET")
```

او إضافة المزيد من العناصر إلى نهاية المجموعة باستخدام الطريقة Add():

```
X.Add ("غادة")
X.Add ("عباس")
X.Add ("برعي")
```

اذا زاد عدد العناصر المضافة (باستخدام الطريقة Add()) عن عدد العناصر الإجمالية (المحددة لحظة تعريف وإنشاء كائن المجموعة) سيتم مضاعفة عدد عناصر المجموعة بشكل تلقائي إلى الضعف، أي ان المجموعة X ستصبح 32 عنصرا والمجموعة Y 200 عنصر.

المزيد أيضا، تحتوي الفئة ArrayList على الطريقتين AddRange() و RemoveRange()، واللتان تمكنانك من إضافة مجموعة إلى نفس المجموعة، او حذف مجموعة عناصر من نفس المجموعة، الإجراء التالي سيدمج كلا المصفوفتين في واحدة:

```
Function JoinTwoArrays(ByVal arr1 As ArrayList, _
    ByVal arr2 As ArrayList) As ArrayList

    JoinTwoArrays = New ArrayList(arr1.Count + arr2.Count)

    JoinTwoArrays.AddRange(arr1)
    JoinTwoArrays.AddRange(arr2)
End Function
```

أخيرا، يمكنك حذف احد عناصر المجموعة باستخدام الطريقة Remove() او الطريقة Clear() ان أردت حذف جميع العناصر بخطوة واحدة (فالمجموعة ArrayList تحتوي على الواجهة IList):

```
X.Remove ("عباس")
Y.Clear ()
```

يستحيل علي ذكر كل التفاصيل وشرح الخصائص والطرق للفئات السابق ذكرها، ولكن كل ما حاولت تقديمه في هذا الفصل هو مدخل إلى مجموعة من الفئات الأساسية التابعة لإطار عمل NET Framework. والتي لا يكاد يخلو أي برنامج منها. حاول إلقاء نظرة إلى مراجع NET. Documentation للحصول على كل التفاصيل الصغيرة والكبيرة حول هذه الفئات. والآن دعنا نستريح قليلا من عرض فئات إطار عمل NET Framework. وننتقل إلى موضوع يتعلق باكتشاف الأخطاء.

PDF

اكتشاف الأخطاء

قرأت مرة في احد مجلات البرمجة الجملة التالية: "إن كانت عملية التنقيح Debugging هي الحل لزوال الأخطاء، فان البرمجة هي سبب حدوث تلك الأخطاء".

سيتمحور حديثي معك في هذا الفصل حول الأخطاء البرمجية وطريقة اكتشافها وتداركها باستخدام مجموعة من الفئات المقدمة من إطار عمل NET. Framework. كما سأخصص قسم كامل يتعلق بأدوات التنقيح التي توفرها لك بيئة Visual Studio .NET.

فكرة عامة

تصنف الأخطاء في لغات البرمجة إلى ثلاثة أقسام هي: أخطاء وقت التصميم، أخطاء وقت التنفيذ، والشوائب. وفيما يلي ذكرها:

أخطاء وقت التصميم

أخطاء وقت التصميم Design Time errors (تسمى أحيانا **بالأخطاء النحوية Syntax Errors**) هي أخطاء تحدث نتيجة لعدم إتباعك للصيغة الصحيحة في كتابة الشفرات المصدرية، كاستدعائك لإجراء غير موجود أو لم يتم تعريفه، أو كتابة حلقة For بدون إغلاقها بـ Next.

هذا النوع من الأخطاء هو أسهلها اكتشافا وتنقيحا، حيث تظهر لك نافذة محرر الشفرات Code Editor خطا متعرجا عند مكان حدوث الخطأ، وبمجرد تحريك مؤشر الفأرة عند ذلك الخطأ ستظهر لك أداة التلميح Tool tip موضحة سبب الخطأ (**شكل 7-1**).

```
Sub Main()
    Dim x As String = 256
```

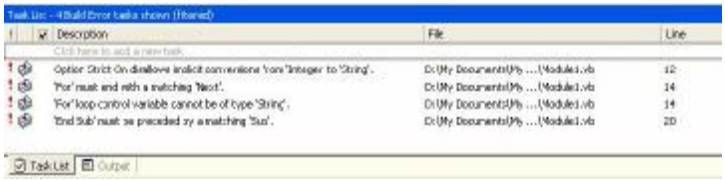
Option Strict On disallows implicit conversions from 'Integer' to 'String'.

شكل 7-1: ظهور أداة التلميح عند الخط المتعرج لتوضيح خطأ وقت التصميم.

إن حاولت ترجمة أو تنفيذ البرنامج، ستظهر لك بيئة التطوير Visual Studio .NET رسالة "There were Build Errors" موضحة بوجود خطأ أو مجموعة اخطأ **(شكل 7-2)**.

شكل 7-2: اكتشاف خطأ أو مجموعة اخطأ نحوية لحظة الترجمة.

إن قمت بالضغط على الزر Yes، سيتم تنفيذ آخر نسخة صحيحة (لا توجد بها أخطاء) من البرنامج، أما نقرك على الزر No سيوقف عملية الترجمة، وستظهر قائمة بكافة الأخطاء النحوية في نافذة Task List **(شكل 7-3)** والتي تصل إليها باختيار الأمر View->Other Windows->Task List.



شكل 7-3: قائمة بكافة الأخطاء النحوية للمشروع الحالي.

الحل الذي يتبعه المبرمجون لمواجهة هذه الأخطاء بسيط جدا ولا يحتاج إلى فلسفة، فكل ما هو مطلوب منك الانتقال إلى السطر الذي وقع فيها الخطأ وتصحيحه.

أخطاء وقت التنفيذ

كمبرمج Visual Basic .NET، انت المسئول الأول والأخير عن أخطاء وقت التنفيذ Run Time Errors، وهي عبارة عن أخطاء تظهر للمستخدم أثناء تنفيذ البرنامج وتوقف عمل البرنامج. **في عالم .NET. تسمى هذه الأخطاء بالاستثناءات Exceptions**.

سبب حدوث الاستثناءات Exceptions في اغلب الأحوال هو عدم توقع المبرمج بالتغيرات الخارجية لبيئة المنصة التي يعمل بها البرنامج، فمثلا في الفصل القادم **الملفات والمجلدات** سترى

كيف يمكنك نقل ملف في الجهاز من مسار إلى اخر باستخدام الطريقة ()Move التابعة للفئة System.IO.File:

```
System.IO.File.Move("D:\File.EXE", "C:\File.EXE")
```

قد يعمل السطر السابق بنجاح في جهازك الشخصي وذلك لوجود الملف D:\File.EXE عندك، ولكن ان قمت بتنفيذه في جهاز اخر لا يحتوي على نسخة من هذا الملف او نفذته مرة اخرى –في جهازك الشخصي- بعد عملية نقل الملف، ستظهر لك بيئة التطوير Visual Studio .NET رسالة استثناء Exception **(شكل 7-4)** تفيدك بان الملف D:\File.EXE غير موجود.

شكل 7-4: رسالة الاستثناء Exception.

كما أخبرتك سابقا –بصيغة اخرى- عليك كمبرمج Visual Basic .NET من اخذ الاستثناء دائما وأبدا في عين الاعتبار، ففي المثال السابق يفترض التحقق من وجود الملف (باستخدام الطريقة ()Exists) قبل القيام بعملية النقل:

```
If System.IO.File.Exists("D:\File.EXE") Then
    System.IO.File.Move("D:\File.EXE", "C:\File.EXE")
End If
```

لا تعتقد انك تستطيع الالتفاف حول الاستثناءات بهذه السهولة، حيث ان الأمر اعقد من ان يتم حله بجملة شرط واحدة، تخيل مثلا ان الملف السابق D:\File.EXE موجود وعلى قيد الحياة، ولكن عليه خاصية القراءة فقط Read Only، لذلك لن تتمكن من نقله وستظهر رسالة الاستثناء مرة اخرى ان لم تتحقق أيضا من قابلية الكتابة باستخدام الطريقة ()GetAttribtues:

```
Imports System.IO

If Not CBool(File.GetAttributes("C:\Test\program.exe") And _
 FileAttributes.ReadOnly )
    If System.IO.File.Exists("D:\File.EXE") Then
        System.IO.File.Move("D:\File.EXE", "C:\File.EXE")
    End If
End If
```

ليس هذا فقط، بل افترض انه موجود ولا توجد عليه خاصية القــراءة فقــط Read Only ولكنه مفتوح من قبل برنامج اخر وعليه خاصية الإقفال، او ان يكون المسار الهدف غيــر قابــل للكتابة (كأقراص CD-ROM مثلا)، او ان المسار الهدف لا توجد به مساحة كافية لنقل الملــف اليه، او... او... وغيرها الكثير من الاحتمالات التي لا يمكنك تداركها بالاعتماد علــى الجمــل الشرطية.

توجد طريقتين في .NET. Visual Basic تمكنك من تدارك الاستثناءات، الطريقة الاولى باستخدام الكائن Exception والطريقة الثانية باستخدام الكائن Err، والتان سأفصل فيهمــا فــي الفقرات القادمة من هذا الفصل.

الشوائب

حتى لو وصلت إلى مرحلة ما بعد الاحتراف في البرمجة، فانه يستحيل عليك كتابــة برنــامج لا يحتوي على **شوائب Bugs**. الشوائب -في لغات البرمجة- هي أخطاء غير متوقعة ينــتج عنهــا تصرف غير متوقع او منطقي للبرنامج، فمثلا الاجراء التالي يقوم بإنشاء عداد يمكنك من وقــف تنفيذ شيفرات البرنامج لفترة من الوقت تحددها بارسال عدد ثواني الانتظار:

```
Sub Sleep(ByVal numOfSeconds As Integer)
    Dim startTime As Double

    startTime = Timer
    Do
        System.Windows.Forms.Application.DoEvents()
    Loop Until Timer - startTime >= numOfSeconds
End Sub
```

قد يعمل الاجراء السابق بطريقة صحيحة في جهازك وفي جهاز غيرك دون اي مــشاكل، ولكن يوجد في الخوارزم السابق شائب Bug برمجي خطير، حيث ان الإجــراء الــسابق يعتمــد اعتماد كلي على ساعة الجهاز، وان قمت بارسال عدد معين من الثواني بحيث يزيد ويتعدى عــن وقت نهاية اليوم (الساعة 23:59:59) سيتم تنفيذ الحلقة إلى مالا نهاية، مما يضطر المستخدم إلى

ايقاف عمل البرنامج باستخدام نافذة Windows Task Manager (التي تصل اليها بالمفاتيح [Alt] + [Ctrl] + [Del]). يمكنك تصحيح هذا الشائب بتعديل شيفرة الإجراء:

```
Sub Sleep(ByVal numOfSeconds As Integer)
    Const NUMOFSEC As Double = 24 * 60 * 60
    Dim startTime As Double

    startTime = Timer
    Do
        System.Windows.Forms.Application.DoEvents()
    Loop Until CBool((Timer + NUMOFSEC - startTime) Mod _
                              NUMOFSEC >= numOfSeconds)
End Sub
```

صحيح اننا قد تمكنا من اكتشاف الشائب في الشيفرة السابقة، ولكن في البرامج الجدية يصبح الوضع اكثر صعوبة وتعقيدا مما هو عليه. ويؤسفني إخبارك بانه لا توجد طريقة او اسلوب متبع لقمع الشوائب من برامجك، ولكن توجد العديد من ادوات التنقيح Debugging التي تستخدم للتقليل من الشوائب (لاحظ اني ذكرت تقليل الشوائب وليس تفاديها) ستجدها في القسم الاخير من هذا الفصل **ادوات التنقيح من بيئة Visual Studio .NET**.

الكائن Exception

جميع لغات NET. تعتمد الكائن Exception لرمي وتفادي الاستثناءات (الأخطاء)، وفي الحقيقة اعتمدت Microsoft هذا الاسلوب لتوحيد طريقة التعامل مع الاستثناءات في جميع لغات البرمجة الموجه إلى إطار عمل NET Framework.

تقنيا، الكائن Exception هو كائن يتم انشائه لحظة وقوع خطأ في شيفرة البرنامج. الاسم الكامل لهذا الكائن هو System.Exception وهو كائن يتم انشائه بمجرد وقوع استثناء في البرنامج.

في معظم الاحوال لن تستخدم هذا الكائن مباشرة، حيث توجد عشرات الكائنات الاخرى والتي تكون خاصة لاستثناءات معينة، فنجد مثلا الاستثناءات DivideByZeroException، OverFlowException و NotFiniteNumberException والخاصة بالاعداد والعمليات الرياضية، ونجد ايضا الاستثناءات FileNotFoundException، DirectoryNotFoundException، EndOfStreamException ...الخ والخاصة بعمليات

التعامل مع الملفات. وغيرها الكثير من الاستثناءات التي يمكنك البحث عنها في مستندات .NET Documentation.

ضع في عين الاعتبار ان جميع الاستثناءات السابق ذكرها مشتقة وراثيا من الفئة System.Exception. لذلك جميع طرق وخصائص الفئة Exception ستكون موجودة ايضا في الفئات الاخرى.

يحتوي الكائن Exception على مجموعة من الخصائص والطرق، وسأكتفي بذكر اهم خاصتين هما Message و StackTrace، كلا الخاصيتين للقراءة فقط ReadOnly وتعودان بقيم حرفية –من النوع String- الاولى تمثل وصف نصي للاستثناء (مثلا: "Attempted to divide by zero.") والثانية تعود بقيمة تمثل الإجراءات التي رمت الاستثناء والإجراءات التي تداركت الاستثناء.

الشيفرة التي سببت في حدوث خطأ يقال انها **ترمي استثناء Throw an exception**، اما الشيفرة التي تتفادى او تتدارك الخطأ فيقال عنها **تتفادى الاستثناء Catch the exception**. سأبدأ بعرض طريقة تفادى الاستثناءات اولا ومن ثم رميها.

تفادي الاستثناءات Catching Exceptions

بدلا من كتابة عشرات الجمل الشرطية في كل سطر تتوقع حدوث استثناء فيه، يمكنك استخدام التركيب Try ... Catch ... End Try. اضف السطور المتوقع حدوث الاستثناء بها اسفل الكلمة المحجوزة Try، ويمكنك كتابة الشيفرات التي ترغب في تنفيذها عند وقوع الاستثناء اسفل كل عبارة Catch:

```
Try
    '
    ' الشيفرة المتوقع حدوث استثناء بها
    '
Catch
    '
    ' الشيفرة التي ستنفذ ان وقع الاستثناء
    '
End Try
```

تطبيقيا، ستستخدم الكائن Exception عند كتابة شيفرة ردة الفعل لحظة وقوع الاستثناء حتى تميز نوع الاستثناء، يمكنك عمل ذلك مع إضافة اسم للمتغير بعد الكلمة Catch:

```
Dim X, Y As Integer
…
…
Try
    ' احداث استثناءات
    X = 10 \ Y
    X = CInt(10 ^ Y)
Catch ex As Exception
    ' كتابة ردة الفعل عند
    ' وقوع الاستثناءات
    If ex.Message = "Attempted to divide by zero." Then

        ArabicConsole.WriteLine("خطأ القسمة على الصفر")

    ElseIf ex.Message = "Arithmetic operation resulted " &
        _ & "in an overflow." Then

        ArabicConsole.WriteLine("خطأ في خروج العدد خارج مجال المتغير")

    Else

        ArabicConsole.WriteLine("خطأ غير معروف")

    End If
End Try
```

الاعتماد على القيمة النصية –كما في المثال السابق- لتحديد نوع الاستثناء طريقة غير محبذة، اذ يفضل استخدام نوع الفئة بكتابة اسمها (راجع مكتبة MSDN لمعرفة اسماء كافة الفئات الاخرى):

```
Dim X, Y As Integer
…
…
Try
    X = 10 \ Y
    X = CInt(10 ^ Y)
Catch ex As DivideByZeroException
    ArabicConsole.WriteLine("استثناء القسمة على الصفر")
Catch ex As OverflowException
    ArabicConsole.WriteLine("استثناء في قيمة العدد خارج مجال المتغير")
Catch ex As Exception
    ArabicConsole.WriteLine("استثناء غير معروف")
End Try
```

PDF

عند وقوع الاستثناء في الشيفرة السابقة، سيتم التحقق من نوع الاستثناء بدئا من اول عبارة Catch حتى النهاية، وان لم تتوافق الفئة مع الاستثناء فسيتم الانتقال إلى الاستثناء الاخير Exception بشكل تلقائي، حيث ان الكائن Exception يمثل اي استثناء تم رميه مهما اختلف نوعه.

اخيرا، يمكنك الخروج من داخل التركيب Try ... Catch ... End Try في اي وقت بكتابة العبارة Exit Try.

ملاحظة

توجد كائنات استثناءات StackOverflowException و OutOfMemoryException لا يوجد وقت أستطيع تحديده لك لوقوعها. ان قمت بتفادي الاستثناءات السابقة، فانصحك القيام بإنهاء البرنامج فورا (اسفل العبارة Catch) حيث ان الوضع لن يكون مستقر ومناسب لمواصلة تنفيذ البرنامج عند وقوع هذه الاستثناءات.

استخدام When:

يمكنك استخدام الكلمة المحجوزة When برفقة العبارة Catch ان أردت إضافة شرط اضافي لعملية قنص الاستثناءات، فبدلا من كتابة جمل الشرط التالية:

```
Dim x As Integer
…
…
Try
    …
    …
Catch ex As Exception
    If x = 0 Then
        …
    ElseIf x = 1 Then
        …
    Else
        …
    End If
End Try
```

تستطيع استخدام When لكتابة اجمل وتصميم ارقى في شيفراتك المصدرية:

```
Dim x As Integer
…
…
Try
    …
    …
Catch ex As Exception When x = 0
    …
Catch ex As Exception When x = 1
    …
Catch ex As Exception
    …
End Try
```

يمكنك الاستفادة من الكلمة المحجوزة When في حالات معينة، منها –على سبيل المثال لا الحصر - تحديد الخطوة او المرحلة التي وقع فيها الاستثناء:

```
Dim StepNum As Integer
…
…
Try
    StepNum = 1
    …
    …
    StepNum = 2
    …
    …
    StepNum = 3
    …
    …
    StepNum = 4
    …
    …

Catch ex As Exception When StepNum = 1
    …
Catch ex As Exception When StepNum = 2
    …
Catch ex As Exception When StepNum = 3
    …
…
…
End Try
```

PDF

استخدام Finally:

يمكنك استخدام الكلمة المحجوزة Finally في التركيب Try … Catch … End Try ان اردت تنفيذ مجموعة من الشيفرات المصدرية دائما، اعني بكلمة دائما –في هذا السياق- هو تنفيذ الشيفرة في حالة وقوع الاستثناء او عدم وقوعه.

```
Try
    '
    ' الشيفرة المتوقع حدوث استثناء بها
    '
Catch
    '
    ' الشيفرة التي ستنفذ ان وقع الاستثناء
    '
Finally
    '
    ' سيتم تنفيذ هذه الشيفرة دائما
    '
End Try
```

اعيد واكرر، الشيفرة التي تقع اسفل الكلمة المحجوزة Finally سيتم تنفيذها دائما حتى لو استخدمت العبارة Exit Try داخل التركيب Try … Catch … End Try و اي عبارات خروج من الإجراء (كـ Exit Sub، Exit Function، Return).

ملاحظة

تستخدم الكلمة المحجوزة Finally كما ذكرت عندما تنوي تنفيذ الشيفرة سواء وقع استثناء او لم يقع، من أمثلة هذه الحالات قطع الاتصالات بقواعد البيانات او اغلاق الملفات.

رمي الاستثناءات Throwing Exceptions

تحدثت في الفقرة السابقة عن طريقة قنص وتدارك الاستثناءات، وفي هذه الفقرة سنقوم بتطبيق العكس اي نرمي الاستثناءات لنسبب الأخطاء. يمكنك رمي الاستثناء في اي سطر من سطور البرنامج باستخدام الامر Throw مع انشاء كائن الاستثناء:

```
' رمي استثناء
Throw New System.IO.FileNotFoundException()
```

PDF

يمكنك تحديد نص الرسالة في الخاصية Message لكائن الاستثناء لحظة الرمي باستخدام الامر Throw:

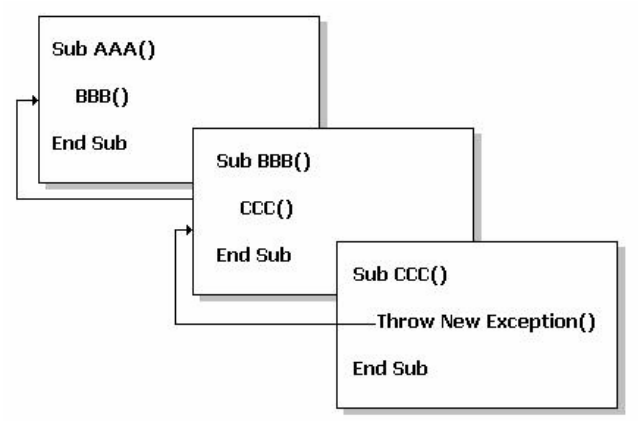
```
Throw New Exception("خطأ في عملية التحديث")
```

قد تستغرب مدى الجدوى من رمي الاستثناءات بنفسك ما دامت معظم فئات إطار عمل NET. Framework ترمي استثناءات بمجرد حدوث أخطاء، ولكنك في حالات كثيرة قد تحتاج إلى رمي استثناءات الأخطاء يدويا بنفسك لتحمي إجراءات فئاتك –مثلا- من ارسال قيم غير صحيحة، او ان رغبتك في رمي استثناءات خاصة ببرنامجك –كما سترى قريبا.

اعود إلى موضوع قنص وتدارك الاستثناءات ولكن في سياق هذه الفقرة (رمي الاستثناءات)، حيث اود ان اخبرك ان الاستثناء لا يختفي ولا يموت ان وقع حتى يقابل اول عملية تفاديه باستخدام التركيب Try … Catch … End Try، وحتى تفهم ماذا اقصد من هذه الكلمات، راقب الشيفرة التالية:

```
Sub AAA()
    BBB()
End Sub

Sub BBB()
    CCC()
End Sub

Sub CCC()
    Throw New Exception()
End Sub
```

PDF



شكل 7-5: سيعود الاستثناء إلى الإجراء المستدعي حتى يجد من يتداركه.

في الإجراء CCC() قمت برمي استثناء جديد، وبمجرد وقوع هذا الاستثناء سيتم الخروج من الإجراء والعودة إلى الإجراء السابق BBB() والبحث عن شيفرة تفادي الاستثناء، وان لم تكتب سيتم الانتقال إلى الإجراء AAA() وان لم توجد فالعودة إلى الإجراء السابق وهكذا ... **(شكل 7-5)** وان لم تتم عملية تدارك الاستثناء، فسيتم ظهور رسالة الخطأ.

جرب القيام بتدارك الاستثناء في الإجراء الاول AAA() ويمكنك معرفة مسار الإجراءات التي يشملها الاستثناء عن طريق الخاصية StackTrace -كما لمحت اليها سابقا:

```
Sub AAA()
    Try
        BBB()
    Catch ex As Exception
        ArabicConsole.WriteLine(ex.StackTrace)
    End Try
End Sub

Sub BBB()
    CCC()
End Sub

Sub CCC()
    Throw New Exception()
End Sub
```

مخرجات الشيفرة السابقة ستكون شبيهه بما يلي:

```
at MyNameSpace.Module1.CCC() in D:\VB.NET\Module1.vb:line 31
at MyNameSpace.Module1.BBB() in D:\VB.NET\Module1.vb:line 28
at MyNameSpace.Module1.AAA() in D:\VB.NET\Module1.vb:line 21
```

اخيرا، عملية رمي الاستثناءات تستهلك الكثير من مصادر النظام فلا تحاول اعتبارها كنوع من الحلويات وتكثر شيفراتك المصدرية منها، فمثلا اجعل الإجراء يعود بقيمة False ان لم يتم تنفيذه بالشكل الصحيح بدلا من التكلفة الزائدة وجعله يرمي استثناء كامل.

إنشاء فئات استثناءات خاصة Custom Exceptions

ينصح دائما برمي فئات الاستثناء المعرفة في إطار عمل Framework .NET. دائما، الا انك قد تحتاج إلى انشاء وتعريف استثناءات خاصة بك يوما من الايام، يمكنك من تعريف فئة استثناء بـ Visual Basic .NET بسهولة شديدة عن طريق تعريف فئة تشتق من الفئة System.ApplicationException:

```
Class UnableToLoadUserFileException
    Inherits System.ApplicationException

    …
    …
End Class
```

ليس هذا فقط، بل يمكنك ايضا إعادة قيادة Overrides خصائص وطرق الفئة القاعدية Exception (وهي التي تم اشتقاق System.ApplicationException وراثيا منها):

```
Class UnableToLoadUserFileException
    Inherits System.ApplicationException

    ' Message اعادة قيادة الخاصية
    Public Overrides ReadOnly Property Message() As String
        Get
            Return "لم اتمكن من تحميل ملف المستخدم"
        End Get
    End Property
End Class
```

PDF

والان يمكنك تفادي هذا الاستثناء باستخدام Try … Catch … End Try او رميه باستخدام Throw في اي وقت –كما تفعل مع باقي الاستثناءات الاخرى:

```
Try
    …
    Throw New UnableToLoadUserFileException()
    …
Catch ex As UnableToLoadUserFileException
    …
    ArabicConsole.WriteLine(ex.Message)
    …
End Try
```

ملاحظة

ان كنت ترغب في رمي كائنات استثناءاتك الخاصة Custom Exceptions خارج المشروع الحالي –او المجمع الحالي لدقة اكثر- عليك جعل فئة الاستثناء قابلة للتسلسل Serializable. الفصل التاسع **التسلسل** Serialization يتحدث بشكل مفصل عن هذا الموضوع.

الكائن Err

تتميز لغة البرمجة Visual Basic .NET عن سائر لغات .NET. الاخرى بوجود الكائن Err الذي يمكنك من تفادي ورمي الاستثناءات. وفي حقيقة الامر، وفرت Microsoft هذا الكائن خصيصا لمبرمجي Visual Basic 6 اعتقادا منهم ان Visual Basic .NET نسخة جديدة من Visual Basic 6->1 (لقد وضحت لك وجهة نظري الشخصية حول هذا الموضوع سابقا في مقدمة هذا الكتاب ولا يوجد داعي لتكرار ما ذكرته).

وكما هو الحال مع الكائن Exception، يمكنك تفادي الاستثناءات ورميها ايضا مع الكائن Err ولكن بصيغة مختلفة.

تفادي الاستثناءات Catching Exceptions

يمكنك الكائن Err من تفادي الاستثناءات بصيغة مختلفة عن التركيب Try … Catch … End Try، وهي استخدام العبارة On Error والتي يكون لها صورتين: الاولى تحدد فيها موقع يتم الانتقال اليه بمجرد حدوث استثناء في الشيفرة المصدرية:

PDF

```
Sub MySub()
    On Error GoTo Errorhandler
    …
    …
    ' الشيفرات المتوقع حدوث استثناء بها
    …
    …
    ' لا تنسى كتابة العبارة التالية حتى
    ' لا يتم تنفيذ السطور التالية دائما
    Exit Sub

Errorhandler:
    ' سيتم الانتقال إلى هذا القسم ان
    ' وقع استثناء
    …
    …
End Sub
```

والثانية تطلب الاستمرار من المترجم حتى ان صادف عشرات الاستثناءات في الشيفرة المصدرية ليتجاهلها وكأن شيئا لم يحدث:

```
Sub MySub()
    On Error Resume Next
    …
    …
    ' الشيفرات المتوقع حدوث استثناء بها
    …
    …
End Sub
```

استخدامك لأحد العبارات السابقة سينقل الاستثناء إلى الكائن Err والذي يحتوي على مجموعة من الطرق والخصائص المتعلقة بهذا الاستثناء، كالخاصية Description والتي تماثل الخاصية Message في الكائن Exception:

```
Sub MySub()
    On Error GoTo Errorhandler
    …
    …
    …
    ' من الجيد الخروج من الإجراء
    ' حتى لا يتم تنفيذ الشيفرة بالاسفل
    Exit Sub

Errorhandler:
    ArabicConsole.WriteLine(Err.Description)
End Sub
```

PDF

ملاحظة

لا يمكنك تفادي الاستثناءات بالصيغتين ... On Error و ... Catch ... Try End Try في نفس الإجراء.

رمي الاستثناءات Throwing Exceptions

رمي الاستثناءات باستخدام الكائن Err يتم بسهولة شديدة، حيث كل ما هو مطلوب منك استدعاء الطريقة ()Raise وإرسال رقم لاستثناء:

```
Err.Raise(11)
```

الرقم 11 السابق يؤدي إلى رمي الاستثناء DivideByZeroException، راجع مكتبة MSDN لمعرفة ارقام الاستثناءات الاخرى، او يمكنك استخدام الرقم 1001 لرمي استثناءات خاصة بك:

```
Err.Raise(1001, , "لم اتمكن من تحميل ملف المستخدم")
```

الاختيار بين Exception و Err

تدارك ورمي الاستثناء بالصيغ السابقة يعود اولا واخيرا على نوع الإجراء الذي امتلأ بالشيفرات المصدرية، ولا استطيع نصحك بأيهما افضل. التعامل مع الكائن Err يعطيك مرونة كبيرة في تدارك الاستثناءات ولكنه يسبب بطء في الشيفرة المصدرية، حيث ان استخدام الصيغة On Error ... في اعلى الإجراء يؤدي إلى تدارك جميع الشيفرات التابعة للإجراء –وليس جزء معين كما تفعل مع الصيغة الاخرى Try ... End Try.

من المهم جدا ان تضع في عين اعتبارك دائما ان تفادي الاستثناءات باستخدام الصيغة On Error ... خاص بلغة البرمجة Visual Basic .NET، وذلك يعني ان لغات برمجة .NET الاخرى لا تتبع هذا الاسلوب وقد يؤثر على سير البرامج. قد لا يعنيك الامر حاليا وذلك لانك لا تنوي تعلم لغة برمجة اخرى، ولكن لا تنسى ان فئاتك قد تستخدم من قبل مبرمجين اخرين يستخدمون لغات اخرى.

الفرق الجوهري بين تفادي الاستثناءات بالصيغة On Error ... والصيغة السابقة Try ... End Try، هو ان في الاولى يتم قتل الاستثناء بمجرد الانتهاء من الإجراء، راقب هذا المثال:

```
Sub AAA()
    Try
        BBB()
    Catch ex As DivideByZeroException
        ' لن يتم تنفيذ هذه الشيفرة
        ArabicConsole.WriteLine(ex.Message)
    End Try
End Sub

Sub BBB()
    On Error GoTo ErrorHandler
    Dim X As Integer = 0

    ' احداث استثناء
    X = 10 \ X

    Exit Sub

ErrorHandler:
    …
    …
End Sub
```

قمت بتفادي الاستثناء في الإجراء AAA() واردت تداركه وطباعة مخرجاته على الشاشة ان وقع، ورغم ان الاستثناء وقع فعلا في الإجراء الاخر BBB() الا ان عملية قنص الاستثناء لم تتم بالشكل المتوقع، حيث ان الشيفرة الموجودة اسفل العبارة Catch في الإجراء AAA() لم تنفذ، وذلك لسبب بسيط وهو ان الاستثناء الذي وقع في الإجراء BBB() قد مات بعد نهاية الإجراء.

الذي يتوجب علينا فعله في هذه الحالة، هو إعادة احياء الاستثناء في الإجراء BBB() –ان وقع- قبل نهاية الإجراء، وذلك ليتم إخبار الإجراء المستدعى AAA() وعمل الازم. يتم ذلك برمي الاستثناء Err.GetException مع الامر Throw او الطريقة Err.Raise():

```
Sub BBB()
    On Error GoTo ErrorHandler
    Dim X As Integer = 0

    X = 10 \ X

    Exit Sub

ErrorHandler:
    Throw Err.GetException
End Sub
```

PDF

يفضل رمي الاستثناء في الحاله السابقة بالامر Throw عوضا عن الطريقة Err.Raise() وذلك لانك قد لا تعلم ما هو الاستثناء الذي قد يقع في الإجراء.

ملاحظة

تلاحظ في المثال السابق أنني استخدمت الصيغة On Error Goto X ولم استخدم أختها On Error Resume Next، وذلك عندما استخدمت الصيغة الثانية لم تنجح معي طريقة رمي الاستثناء إلى الإجراء المستدعي Caller.

أدوات التنقيح من بيئة Visual Studio .NET

لنبتعد قليلا عن كتابة الشيفرات المصدرية، ودعنا نبحر في جولة سريعة حول بيئة التطوير Visual Studio .NET لنتعرف –بشكل سريع- على مجموعة من الأدوات التي تفيدك في عمليات تنقيح برامجك بـ Visual Basic .NET.

أساليب التنفيذ

بشكل تقليدي، يمكنك تنفيذ البرنامج كما فعلنا منذ الصفحة الاولى لهذا الكتاب وذلك بالضغط على المفتاح [F5] او اختيار الامر Start من القائمة Debug. مع العلم ان البرنامج سيعمل داخل بيئة التطوير Visual Studio .NET وما زال تحت سيطرتها. اما ان اردت تنفيذ البرنامج –كبرنامج مستقل عن بيئة التطوير- فاختر الامر Start without Debugging من القائمة Debug السابقة او اضغط على المفاتيح [Ctrl] + [F5]، مع العلم ان خدمات التنقيح لن تكون مدعومة لحظة التنفيذ.

ملاحظة

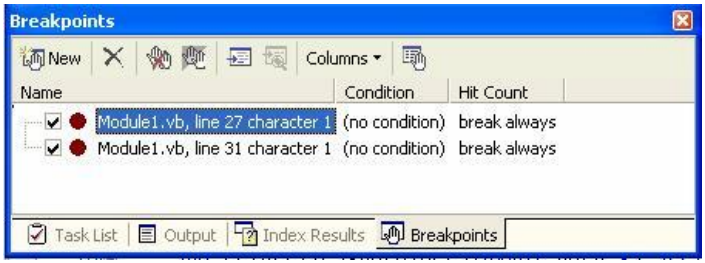
تنفيذ البرنامج باختيار الامر Start without Debugging سيفقد بيئة التطوير السيطرة على البرنامج ولن تتمكن من إيقافه من داخل بيئة التطوير (باختيار الامر Stop Debugging من القائمة Debug)، اي –بعبارة اخرى- سيتم تنفيذ البرنامج بشكل مستقل كما تنفذه من مستكشف النظام Windows Explorer.

يمكنك تحديد **نقاط الوقف Breakpoints** في البرنامج في اي سطر تريده من نافذه محرر الشيفرات المصدرية، وذلك بالضغط بزر الفأرة الايمن على السطر المطلوب واختيار الامر Insert Breakpoint من القائمة المنبثقة، لترى ان السطر قد تغير لونه إلى اللون الاحمر (**شكل 7-6**).

```
7   Module Module1
8
9       Sub Main()
10          Dim x, y, z As Integer
11
12          ArabicConsole.WriteLine(x)
13          ArabicConsole.WriteLine(y)
14          ArabicConsole.WriteLine(z)
15
16
```

شكل 7-6: نقاط الوقف Breakpoints في الشيفرة المصدرية.

يمكنك مشاهدة جميع نقاط الوقف التي وضعتها في شيفراتك المصدرية عن طريق النافذة Breakpoints (**شكل 7-7**) والتي تصل اليها باختيار الامر Debug->Windows->Breakpoints



شكل 7-7: نافذة نقاط الوقف Breakpoints.

لديك اسلوب اخر للتنفيذ يسمى Step Into حيث يتم تنفيذ البرنامج خطوة خطوة، وفي كل مرة تضغط فيها على المفتاح [F11] او تختار الامر Step Into من القائمة السابقة، سيتم تنفيذ سطر واحد فقط من الشيفرات المصدرية (**شكل 7-8**). يفيدك هذا الاسلوب كثيرا ان اردت تتابع الأخطاء ومعرفة مصادرها بين ثنايا الشيفرات المصدرية. وبالنسبة للامر Step Over الموجود

في نفس القائمة Debug، فهو شبيه بالامر Step Into السابق، ولكن التنفيذ سيشمل كامل الإجراء.

```
Dim fileName As String

For Each folderName In Directory.GetDirectories(path)
    ArabicConsole.WriteLine(folderName)
    If subDirectories Then
        PrintFilesAndFolders(folderName, True)
    End If
Next
```

شكل 7-8: التنفيذ سطرا سطرا Step Into.

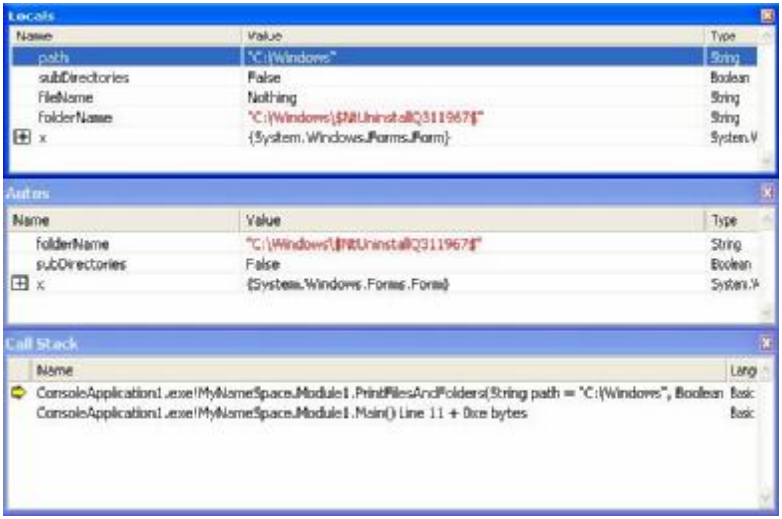
ملاحظة

لن تعمل الاوامر السابقة (Step Into و Step Over) الا اذا كانت الاعدادات Configuration هي Debug وليس Release (**شكل** 1-13 **صفحة** 32). سأعود إلى هذه الاعدادات قريبا جدا.

نوافذ أخرى

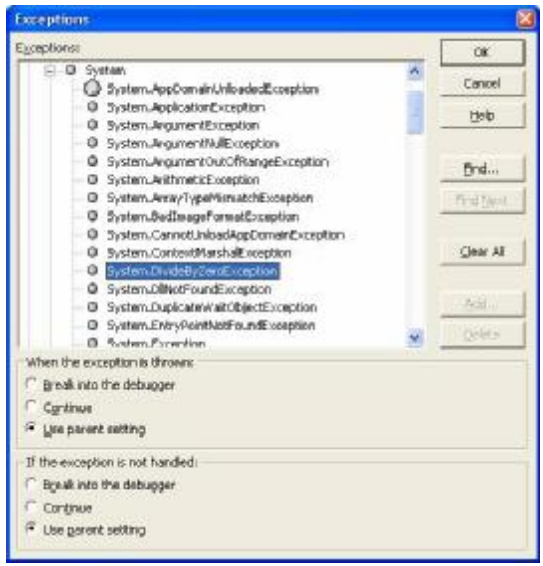
في هذه الفقرة اعرض لك ثلاث نوافذ لن تتمكن من رؤيتها الا لحظة الايقاف المؤقت Pause للبرنامج (والذي قد يكون بسبب نقاط وقف Breakpoints أو التنفيذ Step Into). تفيدك هذه النوافذ في معرفة قيم المتغيرات وطابور تنفيذ الإجراءات. فالنافذة Local تعرض لك قيم المتغيرات المحلية Local Variables للإجراء الحالي، والنافذة Autos تعرض لك قيم المتغيرات التابعة للكائن الذي يتم تنفيذه في السطر الحالي والكائنات في السطور الثلاث السابقة واللاحقة، بينما النافذة Call Stack تعرض لك طابور الإجراءات التي يتم تنفيذها. يمكنك عرض هذه النوافذ لحظة الإيقاف المؤقت باختيار الاوامر المناسبة من القائمة Debug->Windows (**شكل 7-9**).

PDF



شكل 7-9: النوافذ Locals، Autos، و Call Stack.

بعيدا عن النوافذ السابقة، يمكنك ادارة جميع الاستثناءات التي تقع في البرنـــامج او جميـــع استثناءات إطار عمل NET Framework. الاخرى عن طريق صندوق الحـــوار Exceptions والذي تصل اليه باختيار الامر Exceptions من القائمة Debug (**شكل 7-10**).



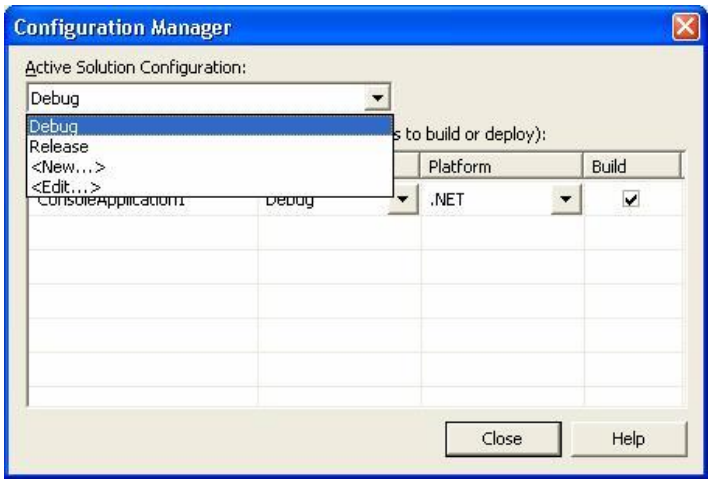
شكل 7-10: صندوق حوار الاستثناءات Exceptions.

الاعدادات Configurations

في الفقرة الاخيرة من الفصل الاول لمحت إلى الاعـــدادات Configurations وذكـــرت انهـــا مجموعة من الخيارات الخاصة بعملية الترجمة Compiling للبرنامج. في كل مـــشروع جديـــد تنشئه، تقوم بيئة التطوير بارفاق اسمين مـــن الاعـــدادات همـــا Debug و Release، تـــستطيع الاختيار بينهما من القائمة الموجودة في شريط الادوات الرئيسي (**شكل 1-13 صفحة 32**).

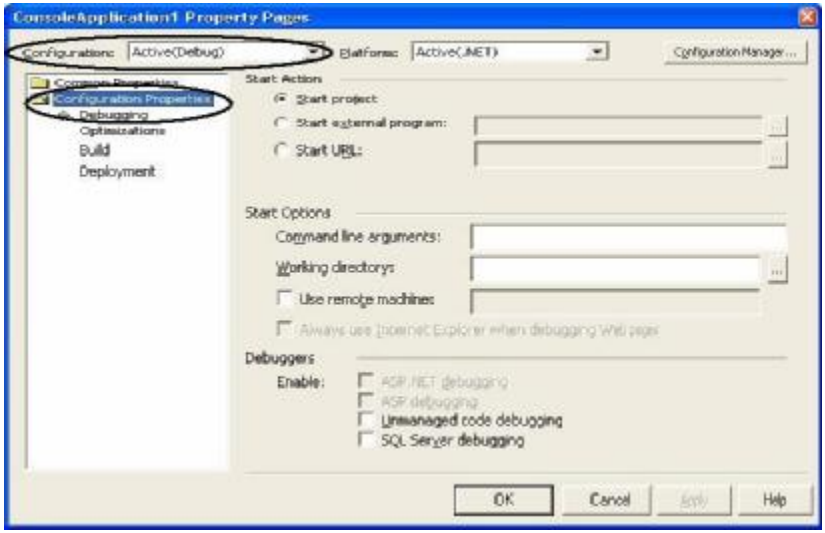
بالاضافة إلى الاعدادات Debug و Release، يمكنك اضافة حذف تعديل اعدادات خاصة بك بالاسماء التي تريدها، يتم ذلك باختيار الامر Configuration Manager من القائمة Build (**شكل 7-11**).

PDF



شكل 7-11: صندوق حوار مدير الاعدادات Configuration Manager.

بعد انشاء اعداد جديد او تعديل الحالي، يمكنك تغيير مواصفات الاعداد وتكوينات الترجمة عن طريق صندوق حوار خصائص المشروع Project Property Pages والانتقال إلى خانة التبويب اليسرى Configuration Properties، مع تحديد الاعداد الحالي في اعلى صندوق الحوار (**شكل 7-12 بالصفحة المقابلة**).



شكل 7-12: تعديل الاعدادات يدويا من صندوق حوار خصائص المشروع.

كان هذا كل ما وددت ذكره حول الأخطاء البرمجية في عالم .NET. من الآن وصاعدا عليك التأقلم على المصطلح الجديد **الاستثناءات Exceptions** حتى لا تسبب لك إرباكا عند قراءتك لمستندات .NET Documentations.. لنغلق ونترك موضوع الاستثناءات جانبا ونتحدث عن موضوع آخر وهو **الملفات والمجلدات** عنوان الفصل الظاهر في بأعلى الصفحة المقابلة.

الملفات والمجلدات

يحتوي مجال الاسماء System.IO من مكتبة فئات إطـــار عمـــل NET. Framework علـــى مجموعة من الفئات التي تمكنك من ادارة وتحرير ملفات ومجلدات الجهاز، بدءاً مـــن العمليـــات البسيطة كالنسخ، النقل، او الحذف، وانتهاءا بتحرير محتويات الملفات برمجيا.

قسمت هذا الفصل القصير الى اربعة اقسام: القسم الأول يتعلق بالمجلدات واستخدام الفئـــة Directory، والثاني بالملفات والتعامل مع الفئة File، والثالث يشرح طريقة تحريـــر الملفـــات ومقدمة الى وحدات التخزين Streams، اما القسم الاخير فيتطرق الى ثلاث فئـــات هـــي Path، DirectoryInfo، و FileInfo بشكل سريع.

ملاحظة

لاختـصار الـشيفرات البرمجيـة، سـأفترض فـي هـذا الفـصل انـك قمـت باسـتيراد مجال الأسماء التالي:

Imports System.IO

الفئة Directory

تحتوي الفئة Directory على مجموعة من الطرق المشتركة Shared Methods تمكنك من تنفيذ وظائف متعددة على المجلدات Folders، كالطريقة CreateDirectory() والتي –من الواضح- انها تنشئ مجلد جديد:

```
Directory.CreateDirectory("C:\Test\My Folder")
Directory.CreateDirectory("C:\المجلد الرئيسي")
```

PDF Factory Pro trial version www.fastio.com

ملاحظة

قبل إجراء اي عملية على الملفات والمجلدات، ينصح بشدة تدارك استثناء دائما:

```
Try
    ...
    Directory.CreateDirectory ("C:\Test")
    ...
Catch
    ...
    ...
End Try
```

على العكس من إنشاء المجلدات، فالطريقة Delete() تحذف مجلد موجود:

```
Directory.Delete("C:\المجلد الرئيسي")
```

ضع في اعتبارك ان السطر السابق لن يعمل الا مع المجلدات الخالية (حيث سيظهر استثناء ان احتوى المجلد على ملفات او مجلدات فرعية)، وان اردت حذف المجلد وكافة المجلدات والملفات التابعة له، ارسل القيمة True مع الوسيطة الثانية للطريقة السابقة:

```
' حذف المجلد بما يحتويه
Directory.Delete("C:\Test", True)
```

ملاحظة

الطريقة Delete() تحذف المجلد –بما يحتويه- بشكل نهائي، فلا تتوقع ارساله الى سلة المهملات Recycle Bin والخاصة بنظام التشغيل.

استخدم الطريقة Move() ان اردت نقل المجلد –بما يحتويه- الى مسار اخر سواء على نفس القرص او إلى قرص اخر:

```
Directory.Move("C:\Test", "C:\المجلد الرئيسي\Test")
```

PDF

يمكنك استخدام الطريقة Move() ايضا لتغيير اسم المجلد دون نقله، وذلك شريطة استخدام نفس المسار:

```
Directory.Move("C:\Test", "C:\TestEx")
```

يفضل دائما التحقق من وجود المجلدات قبل اجراء العمليات السابقة عليها (منعا للشوائب او وقوع الاستثناءات)، يتم ذلك باستدعاء الطريقة Exists() والتي تعود بالقيمة True ان وجد المجلد المرسل:

```
If Directory.Exists("C:\Test") = True Then
    …
    …
End If
```

الطريقة GetCreationTime() تعود بوقت وتاريخ عملية انشاء المجلد، بينما الطريقة GetLastAccessTime() تعود بوقت وتاريخ اخر عملية وصول الى المجلد، اما الطريقة GetLastWriteTime() فتعود بوقت وتاريخ اخر عملية كتابة. مع العلم ان جميع الطرق السابقة تعود بقيمة من النوع Date:

```
ArabicConsole.WriteLine(Directory.GetCreationTime("C:\Test"))
ArabicConsole.WriteLine(Directory.GetLastAccessTime("C:\Test"))
ArabicConsole.WriteLine(Directory.GetLastWriteTime("C:\Test"))
```

يمكنك تعديل القيم السابقة للمجلد باستخدام الطرق SetCreationTime()، SetLastAccessTime()، و SetLastWriteTime():

```
ArabicConsole.WriteLine(Directory.GetCreationTime("C:\Test"))
Directory.SetCreationTime("C:\Test", Date.Now)
ArabicConsole.WriteLine(Directory.GetCreationTime("C:\Test"))
```

طرق تعود بمسارات

الطريقتان GetCurrentDirectory() و GetDirectoryRoot() تعودان بقيمة حرفية (من النوع String)، الاولى تمثل الدليل الحالي، والثانية الدليل الجذري للمسار المرسل:

```
' C:\Test
ArabicConsole.WriteLine(Directory.GetCurrentDirectory)
' C:\
ArabicConsole.WriteLine(Directory.GetDirectoryRoot("C:\Test"))
```

PDF Factory Pro trial version www.fastio.com

وعلى ذكر الطريقة GetCurrentDirectory()، تستطيع تغيير المسار الحالي باستخدام الطريقة SetCurrentDirectory():

```
Directory.SetCurrentDirectory("C:\Test")
```

امـــــــا الطـــــــرق GetLogicalDrives()، GetDirectories()، GetFiles()، و GetFileSystemEntries() فهي تعود بمصفوفة من النوع String تمثل –على التوالي- اسماء محركات الاقراص الموجودة في الجهاز، المجلدات الموجودة في المـــسار المرســـل، الملفـــات الموجودة في المسار المرسل، والمجلدات والملفات الموجودة في المسار المرسل:

```
Dim x As String

For Each x In Directory.GetLogicalDrives
    ArabicConsole.WriteLine(x)
Next

…
…

Dim x() As String = Directory.GetFileSystemEntries("C:\Test", _
    "*.exe")

Dim counter As Integer

For counter = 0 To UBound(x)
    ArabicConsole.WriteLine(x(counter))
Next
```

اخيرا، الطريقة GetParent() تعود بالدليل الأبوي للمسار المرسل، نوع القيمة التي تعود بها هي كائن من الفئة DirectoryInfo، سأتطرق لهذه الفئة في القسم الاخير **فئات اخرى** من هذا الفصل.

البحث عن الملفات والمجلدات

باستخدام الطرق السابق ذكرها، يمكنك الاستعلام والبحث عن كافة اسماء وخـــصائص الملفـــات الموجودة في القرص. طورت الإجراء PrintFilesAndFolders() التالي ليطبع كافـــة اســـماء الملفات والمجلدات الموجودة في المسار المرسل، استخدمت اســـلوب **الاســـتدعاءات التراجعيـــة Recursion** في هذا الاجراء ليشمل السرد الملفات الموجودة في المجلدات الفرعية ايضا. يمكنك تطوير هذا الاجراء ايضا ليعرض لك خصائص ومواصفات الملفات:

```
Sub PrintFilesAndFolders(ByVal path As String, _
    Optional ByVal subDirectories As Boolean = False)

    Dim folderName As String
    Dim fileName As String


    For Each folderName In Directory.GetDirectories(path)
        ArabicConsole.WriteLine(folderName)
        If subDirectories Then
            PrintFilesAndFolders(folderName, True)
        End If
    Next

    For Each fileName In Directory.GetFiles(path)
        ArabicConsole.WriteLine(fileName)
    Next
End Sub
```

يمكنك استدعاء الاجراء السابق للبحث عن الملفات والمجلدات في المسار المرسل، كما تستطيع ارسال القيمة True ان اردت البحث عن كافة محتويات المجلدات الفرعية:

```
' C:\Test عرض محتويات المجلد
PrintFilesAndFolders("C:\Test")

' C:\Test عرض محتويات المجلد
' والمجلدات الفرعية
PrintFilesAndFolders("C:\Test", True)
```

الفئة File

تحتوي الفئة File على العديد من الطرق التي تتعامل مع الملفات بشكل مباشر، وقبل ان اعرض لك هذه الطرق، اود ان انوه هنا الى أن بعض الطرق الموجودة في الفئة السابقة Directory مدعومة ايضا في الفئة File كما أن استخدامها يتم بنفس الطريقة:

```
' حذف ملف
File.Delete ("C:\Test\File.EXE")

' نقل ملف
File.Move("C:\Pic.BMP", "D:\Pic.BMP")

' الحصول على وقت تاريخ انشاء الملف، واخر وصول، واخر تعديل
x = File.GetCreationTime("C:\MyFile.DAT")
x = File.GetLastAccessTime("C:\MyFile.DAT")
x = File.GetLastWriteTime("C:\MyFile.DAT")
```

```
' تعديل القيم السابقة
File.SetCreationTime("C:\MyFile.DAT", Now)
File.SetLastAccessTime("C:\MyFile.DAT", Now)
File.SetLastWriteTime("C:\MyFile.DAT", Now)

' التحقق من وجود الملف
If File.Exists("C:\letter.doc") Then
    …
    …
End If
```

استخدم الطريقة Copy() ان اردت نسخ ملف:

```
File.Copy("C:\program.EXE", "C:\program2.EXE")
```

ستظهر رسالة خطأ وقت التنفيذ ان وجد اسم الملف الهدف، لذلك قد تحتاج الى الكتابة فوق الملف –ان وجد- وذلك بارسال القيمة True مع الطريقة:

```
' سيتم الكتابة فوق الملف الهدف
File.Copy("C:\program.EXE", "C:\program2.EXE", True)
```

امـــا ان اردت الحـــصول علـــى مواصـــفات الملفـــات، فيمكنـــك اســـتدعاء الطريقـــة GetAttributes() والتي تعود بقيمة تركيب من نوع Enum هي FileAttributes:

```
Dim fileAttr As FileAttributes

fileAttr = File.GetAttributes("C:\Test\program.exe")

' هل الملف ارشيف
If CBool(fileAttr And FileAttributes.Archive) Then
    ArabicConsole.WriteLine("ارشيف")
End If

' هل الملف للقراءة فقط
If CBool(fileAttr And FileAttributes.ReadOnly) Then
    ArabicConsole.WriteLine("قراءة فقط")
End If

' هل الملف مخفي
If CBool(fileAttr And FileAttributes.Hidden) Then
    ArabicConsole.WriteLine("مخفي")
End If
…
…
…
```

PDF Factory Pro trial version www.fastio.com

وعلى العكس، فان الطريقة SetAttributes() تمكنك من تعديل خصائص الملف –السابقة- بنفسك:

```
' اجعل الملف للقراءة فقط
File.SetAttributes("C:\program.exe", FileAttributes.ReadOnly)

' اجعل الملف للقراءة فقط ومخفي
File.SetAttributes("C:\program2.exe", FileAttributes.ReadOnly _
    Or FileAttributes.Hidden)

' اضف خاصية الاخفاء لخصائص الملف
File.SetAttributes("C:\program3.exe", _
    File.GetAttributes("C:\program3.exe") Or FileAttributes.Hidden)

' الغ خاصية الاخفاء من خصائص الملف
File.SetAttributes("C:\program4.exe", _
    File.GetAttributes("C:\program4.exe") Xor FileAttributes.Hidden)
```

ملاحظة

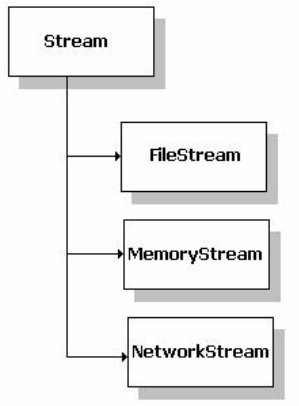
الطريقتان GetAttributes() و SetAttributes() تعملان مع المجلدات Folders ايضا، ولكنك لن تستطيع الوصول اليهما الا عن طريق الفئة File.

الفئة Stream

تمثل الفئة Stream مجموعة من البايتات تكتب وتقرأ من **وحدة تخزين Storage Medium**. قد تكون وحدة التخزين هذه ملف في القرص الصلب File، مقطع من الذاكرة Memory Stream، جهاز افتراضي Virtual Device (كمنفذ متوازي Parallel Port)، او شبكة اتصال Network Stream.

ضع في اعتبارك دائماً ان الفئة Stream هي **فئة مجردة Abstract Class**، لذلك لن تقوم في العادة بانشاء الكائنات منها مباشرة، وانما ستنشئ كائنات من فئات اخرى مشتقة منها **(شكل 8-1)** كالفئات FileStream، MemoryStream، و NetworkStream مع العلم اننا سنتعامل مع الفئة FileStream فقط في هذا الفصل.

PDF



شكل 8-1: العلاقة الوراثية بين فئات وحدات التخزين.

انظر ايضا

قد لمحت سابقا الى الفئات المجردة Abstract Classes في الفصل الرابع **الوراثة**.

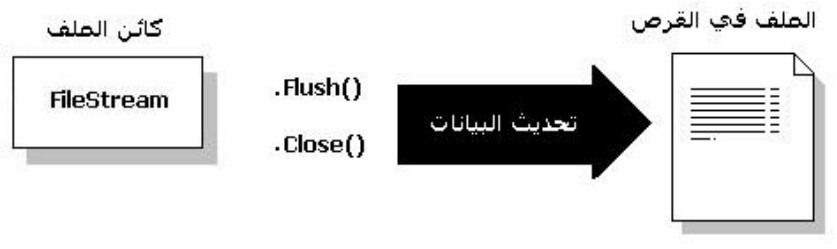
الخصائص والطرق المشتركة

توجد مجموعة من الخصائص والطرق المشتركة بين الفئات المشتقة وراثيا من الفئة Stream باختلاف وحدات التخزين التي تمثلها. وسأبدأ معك بعرض الاعضاء المشتركة، ومن ثم اخصص فقرة لاعضاء الفئة FileStream.

معظم وحدات التخزين تمكنك من الكتابة، القراءة، وتحريك مؤشر الكتابة والقراءة، مع ذلك توجد وحدات تخزين –كالفئة NetworkStream- تسمح لك بالقراءة والكتابة دون امكانية تحريك المؤشر. يمكنك معرفة مدى قدرة الفئة (وحدة التخزين) على اجراء هذه العمليات بالاستعلام عنها في الخصائص CanRead، CanWrite، و CanSeek والتي تعود بالقيمة True ان كانت العملية مدعومة.

بعض الفئات تتبع اسلوب يسمى بالـ **Buffering** عند كل عملية قراءة او كتابة، فعند اجراء عمليات التحرير على الملفات (الفئة FileStream) مثلا، فان عملية التحرير لا تتم على الملف مباشرة بعد تنفيذ الامر، وانما سيتم الاحتفاظ ببيانات الملف في Buffer خاص بها ولن

تتم عملية التحديث الا بعد قيامك باغلاق الملف، او باستدعاء الطريقة Flush() لاجراء التحديثات على الملف (**شكل 8-2**).



شكل 8-2: استدعي الطريقة Flush() او أغلق الملف لتحديث البيانات الى الملف

جميع فئات وحدات التخزين –كما ذكرت- مشتقة وراثيا من الفئة Stream، لذلك فمعظم خصائص وطرق الفئة Stream ستجدها ايضا في الفئات المشتقة، كالخاصية Length التي تعود بحجم البيانات في وحدة التخزين، والخاصية Position التي تحدد موقع مؤشر الكتابة والقراءة في وحدة التخزين.

اما الحديث عن الطرق، فتوجد مجموعة من الطرق كالطريقة Read() التي تعود بمصفوفة من النوع Byte Array تمثل البايتات التي تم قراءتها، والطريقة ReadByte() التي تقرأ بايت واحد فقط. وعلى العكس، الطريقة Write() التي تمكنك من كتابة مجموعة من البايتات في مصفوفة من النوع Byte Array الى وحدة التخزين، والطريقة WriteByte() التي تكتب بايت واحد فقط.

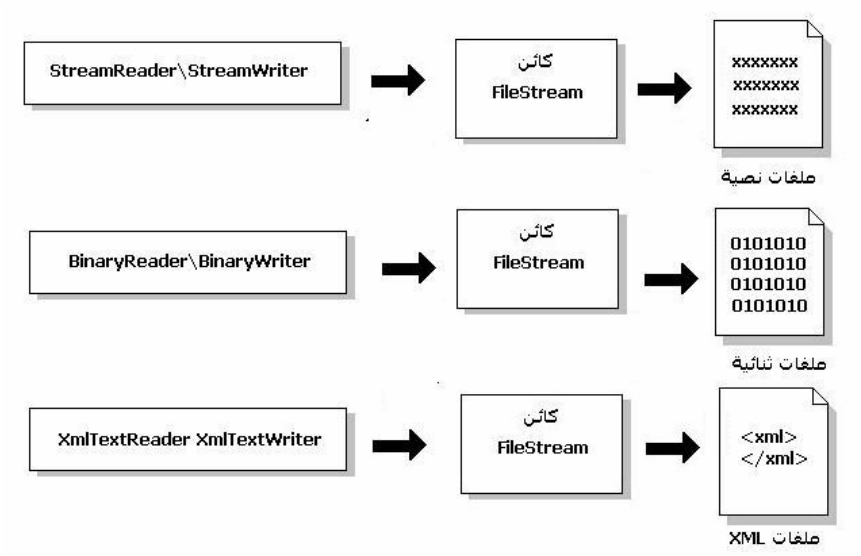
ملاحظة

عملية القراءة والكتابة من والى وحدة التخزين تؤدي الى تغيير موقع مؤشر الكتابة والقراءة في الخاصية Position بشكل تلقائي، بحيث يكون: الموقع الحالي + عدد البايتات التي تم قراءتها او كتابتها.

PDF

المزيد ايضا، الطريقة Close() تغلق وحدة التخزين وتزيل كافة البيانات المتعلقة بها من الذاكرة، وضع في الاعتبار ان اغلاق وحدة التخزين بالطريقة Close() سيستدعي ايضا –بشكل تلقائي- الطريقة Flush() ان كانت وحدة التخزين تعتمد اسلوب الـ Buffering -كالفئة FileStream.

عمليات القراءة والكتابة (باستخدام الطرق السابقة Read()، Write() ... الخ) تتم على شكل بايتات موزعة في مصفوفات من النوع Byte Array، ولكنك قد لا تتبع هذا الاسلوب في اغلب تطبيقاتك، وقد تلجأ الى استخدام مجموعة من الفئات الاخرى التي تسهل عليك عملية القراءة والكتابة (كقراءة او كتابة سطر من نص، قراءة او كتابة مجموعة من البايتات تمثل نوع معين من البيانات كـ Integer، Double ...الخ)، يمكنك عمل ذلك باستخدام الفئات StreamReader، StreamWriter، BinaryReader، BinaryWriter، XmlTextReader، و XmlTextWriter. وهذه الفئات ليست سوى واجهة للمبرمج تستخدم في بنيتها التحتية كائن FileStream للوصول الى الملفات (**شكل 8-3**).



شكل 8-3: أساليب متعددة لاستخدام كائن الملفات FileStream.

نقطة هامة اخيرة (ما أكثر النقاط الهامة والأخيرة في كتابي!): بعد قيامك في كل مرة بانشاء اي كائن –مهما كانت وحدة التخزين والفئة التي تمثله- عليك القيام باغلاقه باستدعاء الطريقة Close() دائما قبل حدوث الموت المنطقي له.

انظر ايضا

سنتعامل في هذا الفصل مع الفئات StreamReader، StreamWriter، BinaryReader، و BinaryWriter. وقد نستخدم الفئتين XmlTextReader و XMLTextWriter لاحقا في الفصل التاسع عشر **ربط البيانات والتكامل مع** XML. المزيد ايضا، توجد فئات اخرى لم استخدمها في هذا الكتاب يمكنك البحث عن تفاصيلها في مكتبة MSDN.

التعامل مع الملفات النصية

عند التعامل مع الملفات النصية Textual Files، فانك ستستخدم الفئة StreamWriter وتنشئ كائن منها لانشاء ملفات نصية والكتابة فيها، لديك العديد من الطرق لانشاء الكائن وفتح الملف النصي والكتابة عليه، وهذه اسهلها:

```
Dim textFile As New StreamWriter("C:\Test.TXT")
```

عند وجود الملف سيتم الكتابة فوقه ويكون موقع مؤشر الكتابه في بداية الملف، اما ان اردت فتح الملف للاضافة Append، فارسل القيمة True الى مشيد الفئة السابقة:

```
Dim textFile As New StreamWriter("C:\Test.TXT", True)
```

المزيد ايضا، تستطيع تحديد نوع صفحة المحارف اما ASCII او Unicode:

```
' ASCII
Dim textFile As New StreamWriter("C:\Test.TXT", True, _
       System.Text.Encoding.ASCII)

' UNICODE
Dim textFile2 As New StreamWriter("C:\Test2.TXT", True, _
       System.Text.Encoding.Unicode)
```

PDF

يمكنك استخدام الفئة FileStream لوضع اعدادات اضافية عند عملية الفتح، كتحديد رغبة انشاء ملف جديد او فتح نفس الملف، تحديد خاصية القراءة والكتابة، وتحديد خاصية المشاركة Sharing (راجع مكتبة MSDN لمزيد من التفاصيل):

```
Dim textStream As FileStream = File.Open("C:\Test.TXT",_
    FileMode.CreateNew, _
    FileAccess.ReadWrite, _
    FileShare.Read)

Dim textFile As New StreamWriter(textStream)
```

بعد فتح الملف، ستبدأ –على الارجح- بالكتابة وارسال البيانات المطلوبة اليه، استخدم الطريقة Write() او الطريقة WriteLine() للكتابة الى الملف، الطريقة Write() ستحول جميع البيانات الى النوع الحرفي String، والطريقة WriteLine() لا تعمل الا مع القيمة الحرفية من النوع String حيث تضيف حرف **سطر جديد NewLine** في نهاية السطر:

```
Dim textFile As New StreamWriter("C:\test.TXT")
…
…
textFile.WriteLine("البيانات المحفوظة")
textFile.Write(99.9)
…
…
textFile.Close()
```

ملاحظة

تذكر ان الطريقة Write() السابقة تابعة للفئة StreamWriter وليس الفئة FileStream. حيث ان الطريقة التابعة للفئة FileStream تتعامل مع المصفوفات من النوع Byte Array فقط.

عملية الكتابة الى الملفات تتبع اسلوب الـ Buffering كما ذكرت في الفقرات السابقة، يمكنك الغاء هذا الاسلوب ان اردت باسناد القيمة True الى الخاصية AutoFlush وستتم عملية التحديث مباشرة بعد كل عملية كتابة الى الملف، اما القيمة False فتمنع عملية التحديث المباشر وسيتم حفظ البيانات بشكل مؤقت في الذاكرة حتى تغلق الملف (باستدعاء الطريقة Close()) او اجراء التحديث يدويا باستدعاء الطريقة Flush().

على العكس من عملية الكتابة، يمكنك قراءة البيانات من الملفات النصية باستخدام الفئة StreamReader. وكما هو الحال مع الفئة StreamWriter السابقة، لديك العديد من الطرق التي تمكنك من فتح ملف للقراءة، وهذه اسهلها:

```
Dim textFile As New StreamWriter("C:\Test.TXT")
```

بعد فتح الملف، فسيكون لديك عدة طرق تمكنك من قراءة البيانات بصور مختلفة. الطريقة ReadLine() تقرأ سطر كامل وتعود بقيمة حرفية من النوع String:

```
Dim textFile As New StreamReader("C:\test.TXT")
…
ArabicConsole.WriteLine( textFile.ReadLine() )
…
textFile.Close()
```

ان كان حجم الملف صغير، فلديك الطريقة ReadToEnd() والتي تقرأ كامل الملف بخطوة واحدة:

```
Dim textFile As New StreamReader("C:\test.TXT")
…
ArabicConsole.WriteLine(textFile.ReadLineToEnd)
…
textFile.Close()
```

ملاحظة

بالنسبة للطريقة Read() فقد تم إعادة تعريفها Overloads بصور مختلفة، لذلك أنصحك بالرجوع الى مكتبة MSDN لمزيد من التفصيلات.

في الامثلة السابقة استخدمت الفئتين StreamWriter و StreamReader للتعامل مع الملفات النصية. أود ان انوه هنا إلى أنك تستطيع الوصول الى الطرق الاضافية لوحدة التخزين (الفئة FileStream) عن طريق الخاصية BaseStream، هنا استخدمت الخاصية Length لمعرفة حجم الملف:

PDF Factory Pro trial version www.fastio.com

```
Dim textFile As New StreamReader("C:\test.TXT")
…
ArabicConsole.WriteLine( textFile.BaseStream.Length )
…
textFile.Close()
```

التعامل مع الملفات الثنائية

ان اردت التعامل مع الملفات الثنائية Binary Files، فيستحسن استخدام الفئات BinaryWriter و BinaryReader. مع ذلك، لن تستطيع فتح الملفات منها مباشرة بكتابة اسم الملف مع المشيد، وانما عليك انشاء نسخة كائن من النوع FileStream وارساله الى مشيدات هذه الفئات:

```
Dim st As FileStream = File.Open("C:\test.dat", FileMode.Create)
Dim binaryFile As New BinaryWriter(st)
```

تستطيع الكتابة الى الملف باستخدام الطريقة Write() والتي تقبل معظم انواع البيانات القياسية (كـ String، Char، Integer، Double … الخ):

```
Dim st As FileStream = File.Open("C:\test.dat", _
    FileMode.OpenOrCreate)

Dim binaryFile As New BinaryWriter(st)

' Integer قيمة من النوع
binaryFile.Write(10)
' Double قيمة من النوع
binaryFile.Write(200.5)
' Boolean قيمة من النوع
binaryFile.Write(True)
…
…
…
binaryFile.Close()
```

وعند القراءة، فستنشئ كائن من الفئة BinaryReader، والتي تحتوي على مجموعة من الطرق بالاسم ReadXXX() حيث تحدد نوع البيانات التي تود قراءتها:

```
Dim st As FileStream = File.Open("C:\test.dat", FileMode.OpenOrCreate)
Dim binaryFile As New BinaryReader(st)

' Integer قيمة من النوع
ArabicConsole.WriteLine(binaryFile.ReadInt32())
' Double قيمة من النوع
ArabicConsole.WriteLine(binaryFile.ReadDouble())
' Boolean قيمة من النوع
ArabicConsole.WriteLine(binaryFile.ReadBoolean())
```

ملاحظة

ان لم تحدد الاستدعاء الصحيح للطريقة ()ReadXXX والذي يتوافق مع النوع المناسب، فستعود الطريقة بقيمة غير صحيحة.

وللتعامل مع البيانات الحرفية Strings فإن ذلك يتطلب بعض التفصيل، حيث ان ارسال قيمة حرفية الى الطريقة ()Write يؤدي إلى كتابة قيمة اضافية قبل الحروف تمثل عدد الحروف التي تضمنتها القيمة الحرفية المرسلة، فالعمليات التالية:

```
Dim st As FileStream = File.Open("C:\test.dat", _
    FileMode.OpenOrCreate)
Dim binaryFile As New BinaryWriter(st)

binaryFile.Write("عباس السريع")
binaryFile.Write("برعي ابو جبهة")
binaryFile.Write("زكريا زعتر")
```

تؤدي الى اضافة حجم الثابت الحرفي قبل الثابت نفسه في الملف، وذلك اشارة الى عدد الحروف التي يتضمنها الثابت حتى تأخذها الطريقة ()ReadString بعين الاعتبار في كل مرة يتم استدعائها:

```
Dim st As FileStream = File.Open("C:\test.dat", _
    FileMode.OpenOrCreate)
Dim binaryFile As New BinaryReader(st)

ArabicConsole.WriteLine(binaryFile.ReadString()) ' عباس السريع
ArabicConsole.WriteLine(binaryFile.ReadString()) ' برعي ابو جبهة
ArabicConsole.WriteLine(binaryFile.ReadString()) ' زكريا زعتر
```

مع ذلك، انصحك بشدة اخذ الحيطة والحذر عند اتباع الاسلوب السابق عند التعامل مع البيانات الحرفية وذلك بسبب كثرة الاخطاء والشوائب البرمجية الناتجة عن نسيان تحريك موقع مؤشر القراءة والكتابة خاصة وان كنت تحركه كثيرا يدويا بنفسك باستخدام الطريقة ()Seek او الخاصية Position.

لديك اسلوب اخر مفضل وهو ارسال مصفوفة من النوع Char Array الى الطريقة ()Write عند الكتابة الى الملف، واستدعاء الطريقة ()ReadChars مع تحديد عدد الحروف التي تود قراءتها من الملف.

PDF Factory Pro trial version www.fastio.com

ملاحظة

كإضافة لثقافتك البرمجية، يسمى الاسلوب الاول **بالحروف مسبقة الحجم** Prefixed-length Strings اما الاسلوب الثاني فيعرف **بالحروف ثابتة الحجم** Fixed-length Strings. اتمنى منك التمييز بين هذه المصطلحات ان رأيتها يوما في مستندات .NET Documentation.

تكوين Custom Streams خاصة

الفئات السابقة التي استخدمناها (StreamReader، StreamWriter، BinaryReader ...الخ) تعمل بشكل رائع مع انواع البيانات القياسية (Integer، String، Boolean، Double وغيرها) ولكن ان حاولت استخدام انواع بيانات خاصة بك كالفئة Person التالية:

```
Class Person
    Public Name As String
    Public Age As Integer
End Class
```

فعليك التعامل مع كل حقل على حدة عند القراءة والكتابة:

```
Dim st As FileStream = File.Open("C:\test.dat", FileMode.OpenOrCreate)
Dim binaryFile As New BinaryWriter(st)
Dim Turki As New Person()

Turki.Name = "تركي العسيري"
Turki.Age = 99

binaryFile.Write(Turki.Name)
binaryFile.Write(Turki.Age)
```

ولو كانت الفئة تحتوي على عشرات الحقول والخصائص، فمن غير العملي القيام بكتابة او قراءة كل حقل على حدة مع الملف.

في مثل هذه الحالات، قد تطور فئات خاصة لتمكنها من التعامل مع انواع بيانات اخرى تعرفها بنفسك. تستطيع عمل ذلك بفضل الوراثة Inheritance، حيث أن كل المطلوب منك هنا هو اشتقاق فئة من فئات الكتابة السابقة وتطبيقها مع الملفات، هذا مثال اشتقت فيه الفئتين BinaryWriter و BinaryReader:

```
' فئة الكتابة
Class PersonWriter
    Inherits BinaryWriter

    Sub New(ByVal st As System.IO.Stream)
        MyBase.New(st)
    End Sub

    Public Overloads Sub Write(ByVal personObject As Person)
        MyBase.Write(personObject.Name)
        MyBase.Write(personObject.Age)
    End Sub
End Class

' فئة القراءة
Class PersonReader
    Inherits BinaryReader

    Sub New(ByVal st As System.IO.Stream)
        MyBase.New(st)
    End Sub

    Function ReadPerson() As Person
        Dim tmpPerson As New Person()

        tmpPerson.Name = MyBase.ReadString
        tmpPerson.Age = MyBase.ReadInt32

        Return tmpPerson
    End Function
End Class
```

وهنا تطبيق عملي يستخدم فئاتنا الجديدة، هذه شيفرة الكتابة الى الملف:

```
Dim st As FileStream = File.Open("C:\test.dat", FileMode.OpenOrCreate)
Dim MyFile As New PersonWriter(st)
Dim Turki As New Person()

Turki.Name = "تركي العسيري"
Turki.Age = 99

MyFile.Write(Turki)

MyFile.Close()
```

وهذه للقراءة من الملف:

```
Dim st As FileStream = File.Open("C:\test.dat", FileMode.OpenOrCreate)
Dim MyFile As New PersonReader(st)
Dim Turki As Person
```

PDF

```
Turki = MyFile.ReadPerson

ArabicConsole.WriteLine(Turki.Name)   ' تركي العسيري
ArabicConsole.WriteLine(Turki.Age)    ' 99

MyFile.Close()
```

ملاحظة

عند اضافة المزيد من الحقول في الفئة Person السابقة، عليك اضافة شيفرات الكتابة الاضافية في الفئة PersonWriter والقراءة في الفئة PersonReader. مع ذلك، سأريك في الفصل القادم **تسلسل الكائنات** Object Serialization كيف يمكن للفئة ان تعلم بجميع حقولها دون الحاجة لتحديثها واضافة المزيد من الشيفرات المصدرية والخاصة بالكتابة او القراءة.

فئات أخرى

قبل ان اختم هذا الفصل، بودي ذكر ثلاث فئات ستسهل عليك الكثير من المهام المتعلقة بالملفات والمجلدات ان احسنت استخدامها بذكاء.

الفئة Path

تحتوي الفئة Path على مجموعة من الطرق والخصائص التي تتعلق بحروف المسارات Paths المستخدمة في الملفات والمجلدات. خذ مثلا هذه الخصائص التي تعود بالحروف المستخدمة في تحديد فواصل المسارات (قد تختلف النتائج عند تطبيق الشيفرة التالية على انظمة تشغيل اخرى غير Windows):

```
' /
ArabicConsole.WriteLine(Path.AltDirectorySeparatorChar)
' \
ArabicConsole.WriteLine(Path.DirectorySeparatorChar)
' "<>|
ArabicConsole.WriteLine(Path.InvalidPathChars)
' ;
ArabicConsole.WriteLine(Path.PathSeparator)
' :
ArabicConsole.WriteLine(Path.VolumeSeparatorChar)
```

المزيد ايضا، هذه مجموعة اضافية من الطرق تمكنك من استخلاص جـزءاً معينـاً مـن المسار، كاسم الملف، المجلد، الامتداد وغيرها:

```
Dim X As String = "C:\Test\File.EXE"

' C:\Test
ArabicConsole.WriteLine(Path.GetDirectoryName(X))
' File.EXE
ArabicConsole.WriteLine(Path.GetFileName(X))
' .EXE
ArabicConsole.WriteLine(Path.GetExtension(X))
' C:\
ArabicConsole.WriteLine(Path.GetPathRoot(X))
' True
ArabicConsole.WriteLine(Path.HasExtension(X))
' File
ArabicConsole.WriteLine(Path.GetFileNameWithoutExtension(X))
```

الفئات DirectoryInfo و FileInfo

تمثل الفئة DirectoryInfo مجلد معين، والفئة FileInfo ملف معين، لانشاء كائنات مـن هـذه الفئات، ارسل قيمة حرفية تمثل مسار المجلد او الملف مع مشيد الفئة:

```
' مجلد
Dim folder As New DirectoryInfo("C:\Windows")
' ملف
Dim file As New FileInfo("C:\Autoexec.bat")
```

كلا الفئتان مشتقتان وراثيا من الفئة FileSystemInfo وبالتالي فإنهما تحويان مجموعة من الطرق والخصائص المشتركة مثل: Name، FullName، Extension، Exists، Attributes، CreationTime، LastWriteTime، LastAccessTime، Refresh()، و Delete():

```
' Windows
ArabicConsole.WriteLine(folder.Name)
' Autoexec.bat
ArabicConsole.WriteLine(file.Name)

' C:\Windows
ArabicConsole.WriteLine(folder.FullName)
' C:\Autoexec.bat
ArabicConsole.WriteLine(file.FullName)
…
…
```

تحتوي الفئة DirectoryInfo على مجموعة من الطرق والخصائص الخاصة بها، كالخاصية Parent التي تعود بالمجلد الأبوي للمجلد الحالي، والطريقة Create() التي تنشئ مجلد جديد، ومجموعة من الطرق الاخرى. هذا مثال يشرح استخدام الطريقة GetDirectories() التي تعود بكائنات (من النوع DirectoryInfo) تمثل المجلدات الفرعية التابعة للمجلد الذي يمثله الكائن:

```
Dim folder As New DirectoryInfo("C:\Windows")
Dim subfolder As DirectoryInfo

For Each subfolder In folder.GetDirectories
    ArabicConsole.WriteLine(subfolder.Name)
Next
```

اما الفئة FileInfo فهي ايضا تحتوي على مجموعة من الطرق والخصائص الخاصة بطبيعة عملها كالخاصية Length التي تعود بحجم الملف، ومجموعة من الطرق الاضافية التي تستخدم لفتح وانشاء الملفات. هذا مثال اخر شبيه بالمثال السابق، ولكنه يستدعي الطريقة GetFiles() (التابعة للفئة DirectoryInfo) للعودة بكائنات من النوع FileInfo:

```
Dim folder As New DirectoryInfo("C:\Windows")
Dim subfile As FileInfo

For Each subfile In folder.GetFiles("*.EXE")
    ArabicConsole.WriteLine(subfile.Name)
Next
```

ملاحظة

يمكنك تطوير الشيفرتين السابقتين لتعرض أحجام ومواصفات الملفات وغيرها من معلومات، وذلك باستخدام خصائص كائنات الحلقة subfolder و subfile.

PDF

في هذا الفصل تعرفنا على مجموعة من الفئات والخاصة بالتعامل مع الملفات والمجلدات، وقد تطرقت فيه الى وحدات التخزين Streams وطريقة عملها، وفي الحقيقة كان لدي هدف اخر من هذا الفصل وهو تمهيدك الى استخدام وحدات التخزين Streams لتطبيق ما يعرف **بتسلسل الكائنات Object Serialization** عنوان الفصل التالي.

PDF

تسلسل الكائنات Object Serialization

تعلمت في الفصل الثالث **الفئات والكائنات** طريقة انشاء نسخة Instance من الكائن، وكانت العلاقة التي تربطك بهذا الكائن محصورة فقط في مؤشر ذلك الكائن والمتمثلة في متغير داخل شيفرة البرنامج. يمكنك السيطرة على نسخة الكائن Object Instance اكثر والتحكم فيه، وكذلك نقله من مكان الى اخر باستخدام **التسلسل Serialization**.

هذا الفصل هو مدخلك المبدئي الى تسلسل الكائنات Object Serialization، ودعني انوه هنا إلى أن موضوع التسلسل من المواضيع المتقدمة جدا في برمجة اطار عمل NET. Framework، وبالتالي سأفترض انك مبرمج متمكن جدا، حيث ان هذا الموضوع موجه الى المبرمجين الجادين.

ملاحظة

معظم فئات هذا الفصل تجدها في مجالات الاسماء التالية:

```
Imports System.Runtime.Serialization
Imports System.Runtime.Serialization.Formatters.Binary
```

لذلك لا تغفل عن استيرادها في ملفات مشروعك، كما سأستخدم ايضاً مجموعة من فئات الفصل السابق والموجودة في مجال الاسماء التالي:

```
Imports System.IO
```

PDF Factory Pro trial version www.fastio.com

مدخلك إلى تسلسل الكائنات

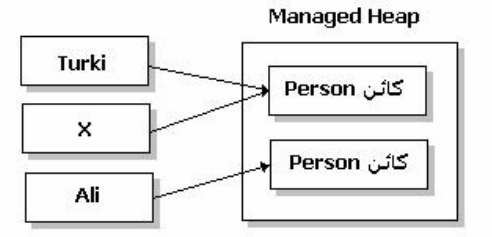
سأبدأ معك هذا الفصل بهذا القسم الذي يعتبر مدخلا الى التسلسل، حيث سنبدأ بتعريف عملية التسلسل في عالم NET. ثم نقوم بتطبيقه بشكل عملي وذلك بعرض مثال بسيط يستخدم مفهوم التسلسل لحفظ الكائن في ملف على القرص الصلب واستعادته مرة أخرى، يلي ذلك عرض لكيفية تطبيق التسلسل على الفئات التي تعرفها بنفسك، وذلك باستخدام بعض المواصفات Attributes التي تجعل الفئة قابلة او غير قابلة للتسلسل.

ما هو التسلسل؟

نظراً للارتباط الوثيق بين التسلسل والكائنات فاسمح لي أن اذكرك بالمقصود من عبارة **نسخة الكائن Object Instance**. نسخة الكائن هي تلك المنطقة من الذاكرة التي تحمل بيانات كائن تم انشائه من فئة، وهي لا تمثل متغيرات الكائن، اذ ان نسخة الكائن قد يشير لها اكثر من متغير، ففي الشيفرة التالية:

```
Dim Turki As New Person
Dim Ali As New Person
Dim X As Person
…
…
X = Turki
```

أنشأت نسختين من الكائن Two object instances رغم وجود ثلاث مؤشرات، حيث ان المؤشرين Turki و X يشيران الى نفس نسخة الكائن Both *Turki* and *X* pointers point to the same object instance (معذرة على كتابتها باللغة الإنجليزية ولكن يهمني جدا استيعابها حتى لو كلفني الامر تعلم اللغة الصينية) **(شكل 9-1)**.



شكل 9-1: ثلاث مؤشرات تشير الى نسختين كائن من الفئة Person.

التسلسل Serialization هي عملية حفظ بيانات نسخة الكائن Object Instance في وحدة تخزين Stream (قد تكون وحدة التخزين هذه ملف في القرص الصلب، مقطع من الذاكرة، حقل في جدول داخل قاعدة بيانات ...الخ)، ويمكنك لاحقا اعادة احياء الكائن وارفاقه بمؤشر في شيفرتك وذلك **بعكس عملية التسلسل Deserializing**.

من الان وصاعدا –وبفضل التسلسل- لن تخشى على كائناتك من الموت أو الضياع، اذ ان بقاء الكائن على قيد الحياة لن يعتمد بعد الآن على خروج مؤشر الكائن من مداه Scope سواءً كان ذلك بنهاية الإجراء، أو بالإنهاء الكلي للبرنامج.

يلعب التسلسل دورا حيويا في تطبيقات إطار عمل NET. Framework، ولا يقتصر استخدامه على حفظ الكائنات في وحدات تخزين كملفات في الاقراص، بل أنه يتجاوز ذلك بمراحل، اذ أن هذه التقنية تمكن المبرمجين من تبادل كائناتهم بين البرامج المختلفة، او حتى بين دول العالم المتباعدة عن طريق شبكة الانترنت باستخدام خدمات Web Services او تطبيقات ASP.NET. وقد اشرت في الفصل السابع **اكتشاف الأخطاء** إلى ان الاستثناءات الخاصة Custom Exceptions يجب ان تكون قابلة للتسلسل Serializable لتتمكن من رميها بين المجمعات المختلفة.

إن جعل الفئة قابلة للتسلسل سيوفر عليك الكثير من الجهد والوقت، حيث أنها ستمكنك من حفظ بيانات الفئات دون الحاجة لكتابة سطور اضافية عند اضافة حقول او خصائص للفئة (كما فعلنا في الفصل السابق **الملفات والمجلدات** عندما طورنا Custom Streams خاصة).

التسلسل بالصيغة الثنائية Binary Serialization

في البداية دعني أطمئنك إلى ان إطار عمل NET. Framework يعرف جيدا كيف يجري التسلسل على انواع البيانات الأولية Primitive Types (كـ String، Integer، Double، Boolean ...الخ)، فكل ما تحتاجه لتطبيق التسلسل على هذه الانواع هو كائن Formatter.

الكائن Formatter هو كائن يحتوي على الواجهة IFormatter (وهي واجهة موجودة في مجال الاسماء System.Runtime.Serialization)، يمكنك تطوير فئات خاصة بك تشتمل على هذه الواجهة، رغم ان إطار عمل NET. Framework يحتوي على مجموعة جيدة من الفئات التي تستطيع استخدامها مباشرة، من هذه الفئات الفئة BinaryFormatter (تجدها في مجال الاسماء System.Runtime.Serialization.Formatters.Binary) وهي فئة تستخدم لحفظ بيانات الكائن بالصيغة الثنائية Binary Format.

سأعرض لك الان مثالا يطبق التسلسل على البيانات القياسية، وفي الفقرات التالية سنطبقه على بيانات خاصة تعرفها على شكل فئات خاصة بك. كل المطلوب منك هو انشاء كائن من الفئة BinaryFormatter واستدعاء الطريقة Serialize() لحفظ بيانات الكائن، تتطلب هذه الطريقة وسيطتين الاولى وحدة التخزين Stream (استخدمت كائن من النوع FileStream في هذا المثال)، والثانية هو الكائن المطلوب تطبيق التسلسل عليه:

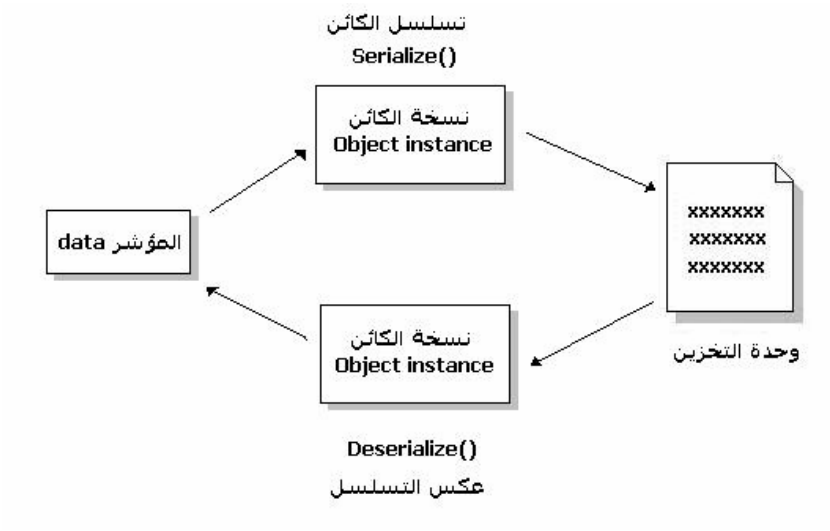
```
Dim data As String() = {"حسونة", "برعي", "عباس"}
Dim st As FileStream = File.Open("C:\test.dat", FileMode.OpenOrCreate)
Dim SerialObj As New BinaryFormatter()

' بدء عملية التسلسل
SerialObj.Serialize(st, data)


' Stream اغلق وحدة التخزين
st.Close()
```

والان تستطيع في اي وقت وباستخدام اي برنامج ومن اي مكان على وجه الارض إعادة إحياء الكائن من جديد، يتم ذلك باستخدام نفس الفئة BinaryFormatter ولكن باستدعاء الطريقة Deserialize() هذه المرة لعكس عملية التسلسل، تتطلب هذه الطريقة وسيطة واحدة فقط وهي وحدة التخزين Stream، وستعود الطريقة بكائن حي يرزق من النوع Object (عليك استخدام المعامل CType لإجراء عملية التحويل المناسبة وذلك بسبب العبارة Option Strict On):

```
Dim data As String()
Dim st As FileStream = File.Open("C:\test.dat", FileMode.OpenOrCreate)
Dim SerialObj As New BinaryFormatter()

' Deserializing عكس عملية التسلسل
data = CType(SerialObj.Deserialize(st), String())

ArabicConsole.WriteLine(data(0))      ' حسونة
ArabicConsole.WriteLine(data(1))      ' برعي
ArabicConsole.WriteLine(data(2))      ' عباس
st.Close()
```

ان ما فعلناه في الشيفرتين السابقتين ليس سوى نقل نسخة بيانات الكائن Object Instance والتي يشير لها المؤشر data الى وحدة تخزين باستدعاء الطريقة Serialize()، ومن ثم قمنا

بعكس العملية ونقلنا نسخة الكائن من وحدة التخزين الـى المؤشـر data باسـتدعاء الطريقـة Deserialize() والخاصة بكائن التسلسل (**شكل 9-2**).



شكل 9-2: تسلسل/عكس تسلسل نسخة الكائن.

والان دعنا نرى كيف يمكننا تطبيق التسلسل على البيانات الاخرى والتي تعرفهـا بنفـسك باستخدام التركيب Class … End Class كما في الفقرة التالية.

تسلسل أنواع بيانات مخصصة (غير قياسية)

تطبيقيا، كل ما تحتاجه لجعل فئاتك قابلـة للتسلـسل هـو اسـتخدام المواصـفة Serializable Attribute، وجعلها في اعلى شيفرة تعريف الفئة فقط:

```
<Serializable()> _
Class Person
    …
    …
End Class
```

ما يجب عليك معرفته هو ان جميع الحقول التابعة للفئة (حتـــى وان كانـــت خاصـــة Private) والمتغيرات الاستاتيكية Static Variables داخل إجراءات الفئة سيطبق عليها التسلسل، ولكـــي تتحقق من ذلك اضف هذه الشيفرة بين فكي التركيب Class ... End Class:

```
<Serializable()> _
Class Person
    Public Name As String      ' Public متغير عام

    Private m_Age As Integer   ' Private متغير خاص
    Property Age() As Integer
        Get
            Return m_Age
        End Get
        Set(ByVal Value As Integer)
            m_Age = Value
        End Set
    End Property

    Function GetValue() As Integer
        Static x As Integer    ' Static متغير ستاتيكي

        x += 1
        Return x
    End Function
End Class
```

والآن ابدأ باستخدام الفئة السابقة، واسند أي قيم لحقولها وخصائصها ولا تنـــسى اســـتدعاء الطريقة GetValue() عدة مرات حتى تزيد من قيمة المتغير الستاتيكي المحفوظ بها، ومن ثم قم بتطبيق التسلسل على الكائن:

```
Dim st As FileStream = File.Open("C:\test.dat", FileMode.OpenOrCreate)
Dim SerialObj As New BinaryFormatter()
Dim Turki As New Person()


Turki.Name = "تـركـي الـعسيري"
Turki.Age = 99
Turki.GetValue()
Turki.GetValue()
Turki.GetValue()

SerialObj.Serialize(st, Turki)

st.Close()
```

ان اردت عكس عملية التسلسل، فلا داعي لانشاء نسخة جديدة من الكائن باستخدام الكلمة المحجوزة New، اذ ان النسخة موجودة وجاهزة في وحدة التخزين (الملف) وسنقوم بإسنادها الى مؤشر (وهو Turki كما في الشيفرة التالية):

```
Dim st As FileStream = File.Open("C:\test.dat", FileMode.OpenOrCreate)
Dim SerialObj As New BinaryFormatter()
Dim Turki As Person

Turki = CType(SerialObj.Deserialize(st), Person)

ArabicConsole.WriteLine(Turki.Name)          ' تركي العسيري
ArabicConsole.WriteLine(Turki.Age)           ' 99
ArabicConsole.WriteLine(Turki.GetValue)      ' 4

st.Close()
```

ملاحظة

عند عكس عملية التسلسل لفئاتك الخاصة، فلابد من ان تكون الفئة المراد عكس تسلسلها معرفة في البرنامج الذي يعيد كائنها، وذلك اما بتعريف الفئة في شيفرة البرنامج، او اضافتها من قائمة المراجع Reference.

من كان يصدق ان بيانات الكائنات يتم حفظها واسترجاعها بهذه السهولة؟ لدي اضافة بسيطة بودي توضحيها عند استخدام المواصفة Serializable Attributes: هذه المواصفة ستجعل جميع بيانات الكائن –كما قلت- قابلة للتسلسل، ان لم ترغب في أن تكون جميع عناصر الفئة قابلة للتسلسل، كأن تكون بعض الخصائص القابلة للعودة بقيمة غير محفوظة (مثل الوقت الحالي او استخلاص قيمة من خصائص اخرى)، او أن بعض الحقول تابعة لفئات غير قابلة للتسلسل اصلاً، ومهما يكن هدفك من تطبيق مفهوم التسلسل، يمكنك منع المتغير من التسلسل ضمن عناصر الفئة باستخدام المواصفة NonSerialized Attributes:

```
<Serializable()> _
Class Person
    Public Name As String
    ' لن يتم تسلسل هذا المتغير
    <NonSerialized()> Private MotherName As String
    Private m_Age As Integer
    …
    …
End Class
```

PDF

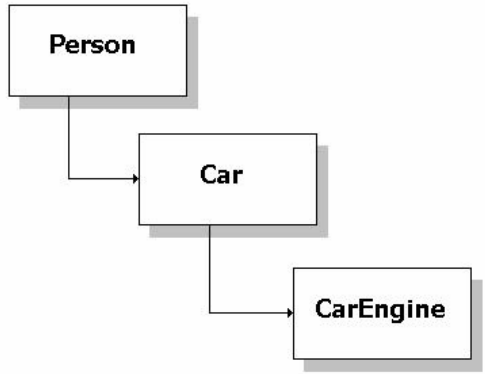
خريطة الكائنات Object Graph

في لغات البرمجة السابقة، لم تخلو برامجنا ومشاريعنا من **الاهرام الكائنية Object Hierarchies** والتي تنجز بفضل علاقة **الاحتواء Containment** بين الفئات، والتي تسمى - في عالم OOP - بعلاقة **يحتوي على Has a**، فما بالك بمشاريعك التي تنجز تحت اطار عمل .NET Framework. والتي يعد كل شيء فيها عبارة عن كائن Object؟

خريطة الكائنات Object Graph هي العلاقات Relationships والمراجع References الموجودة بين الفئات في الهرم الكائني Object Hierarchy، فلو كانت لدينا هذه الفئات الثلاث:

```
<Serializable()> _
Class CarEngine
    Public Cylinder As String
    Public Volume As String
End Class

<Serializable()> _
Class Car
    Public Name As String
    Public Model As String
    Public Engine As New CarEngine()
End Class

<Serializable()> _
Class Person
    Public Name As String
    Public Car As New Car()
End Class
```

فيمكن رسم خريطة الكائنات Object Graph لهذه الفئات كما **(بالشكل 9-3 بالصفحة المقابلة)**. وعليك معرفة ان الشكل يبين علاقة الاحتواء Containment بين الفئات وليس الاشتقاق الوراثي Inheritance.



شكل 9-3: علاقة الاحتواء بين الفئات.

ملاحظة

خريطة الكائنات Object Graph تشمل الكائنات من النوع المرجعي Reference Type فقط، اما البيانات ذات القيمة Value Type فهي تتبع الكائن نفسه، ولا يقال عليها العلاقة **يحتوي على** Has a.

الذي اريد ان اصل إليه معك من الكلام السابق، هو ان تطبيق التسلسل على كائن ما سيؤدي إلى تضمين جميع الكائنات الموجودة في خريطة الكائنات Object Graph لهذا الكائن، وسيطبق التسلسل عليها جميعاً، وهذا هو الدليل:

```
Dim st As FileStream = File.Open("C:\test.dat", FileMode.OpenOrCreate)
Dim SerialObj As New BinaryFormatter()
Dim Turki As New Person()

' اسناد قيم للحقول
Turki.Name = "تركي العسيري"
Turki.Car.Name = "BMW"
Turki.Car.Model = "Class 7"
Turki.Car.Engine.Cylinder = "12 V"
Turki.Car.Engine.Volume = "999 xm"

' تطبيق التسلسل
SerialObj.Serialize(st, Turki)
```

```
' مؤشر للاختبار
Dim Test As Person

' لابد من تحريك مؤشر الملف الى البداية
st.Seek(0, SeekOrigin.Begin)

' عكس التسلسل
Test = CType(SerialObj.Deserialize(st), Person)

' طباعة محتويات الكائن وباقي خريطة الكائنات
ArabicConsole.WriteLine(Test.Name)                    ' تركي العسيري
ArabicConsole.WriteLine(Test.Car.Name)                ' BMW
ArabicConsole.WriteLine(Test.Car.Engine.Cylinder)     ' 12 V
…
…
…

st.Close()
```

حالة المرجعية الدائرية Circular Reference:

ان كنت يا عزيزي القارئ قد قرأت الفصل الثالث **الفئات والكائنات** بتركيز وتمعن، فبلا شك ستتذكر ان المرجعية الدائرية ما هي الى علاقة بين كائنين يشيران الى بعضهما البعض، فلو كان للفئة Person حقل من نفس نوع الفئة:

```
<Serializable()> _
Class Person
    Public Name As String
    Public Brother As Person
End Class
```

وقمت بانشاء كائنين وأحدثت مرجعية دائرية بينهما:

```
Dim Abbas As New Person()
Dim Burey As New Person()

Abbas.Name = "عباس"
Burey.Name = "برعي"
Abbas.Brother = Burey
Burey.Brother = Abbas
```

ثم اردت تطبيق التسلسل على الكائن الاول Abbas، فاستنادا الى قاعدة خريطة الكائنات Object Graph التي تحدثت عنها قبل قليل، سيتم تطبيق التسلسل على الكائن الاول والبحث عن جميع العلاقات التي يحتويها، وهي العلاقة المتمثلة في الحقل Brother، ليشمل الكائن الثاني Burey

ويبحث عن جميع العلاقات التي يحتويها وهي العلاقة Brother، ويعود مرة اخرى الى الكائن الاول Abbas، ومن ثم الى Burey وهكذا الى مالا نهاية.

السيناريو السابق صحيح نظرياً، ولكن يسرني اخبارك بان مطوري فئات التسلسل لديهم من الخبرة البرمجية ما يكفي لتجاوز هذه المشكلة، حيث ان فئات التسلسل المقدمة من اطار عمل NET. لا تقوم بتسلسل الكائن اكثر من مرة، مما يعني ان الكائنين السابقين Abbas و Burey سيتم تسلسلهما مرة واحدة فقط، والشيفرة التالية خير دليل:

```
Dim st As FileStream = File.Open("C:\test.dat", FileMode.OpenOrCreate)
Dim SerialObj As New BinaryFormatter()
Dim Abbas As New Person()
Dim Burey As New Person()

Abbas.Name = "عباس"
Burey.Name = "برعي"
Abbas.Brother = Burey
Burey.Brother = Abbas

' لاتوجد اي مشاكل في عملية التسلسل
SerialObj.Serialize(st, Abbas)


Dim Test As Person

st.Seek(0, SeekOrigin.Begin)
Test = CType(SerialObj.Deserialize(st), Person)
ArabicConsole.WriteLine(Test.Name)            ' عباس
ArabicConsole.WriteLine(Test.Brother.Name)    ' برعي
st.Close()
```

نسخ الكائنات

في الفصل الخامس **الواجهات، التفويض، والمواصفات** عرضت عليك الواجهة ICloneable وذكرت لك أن بامكانك تضمينها في فئاتك ان اردت اعطاء امكانية نسخ كائناتها، وذلك باستدعاء الطريقة MemberwiseClone() التابعة للفئة القاعدية Object. مع ذلك، توجد مشكلة لم اخبرك بها (لان حلها يتم باستخدام التسلسل) وقد حان وقت مناقشتها الآن، وتتمثل في ان الكائنات الاخرى التي تحتويها الفئة لا يمكن نسخها، راقب هذه الفئات:

```
Class Car
    Public Name As String
    Public Model As String
End Class
```

PDF Factory Pro trial version www.fastio.com

```
Class Person
    Implements ICloneable


    Public Name As String
    Public Car As New Car()

    Private Function PrivateClone() As Object Implements
ICloneable.Clone
        Return Me.Clone()
    End Function

    Public Function Clone() As Person
        Return CType(Me.MemberwiseClone(), Person)
    End Function
End Class
```

حسنا، دعنا نجرب نسخ الكائن ونرى ما اذا كانت الكائنات المحضونة فيه سيتم نسخها ام لا:

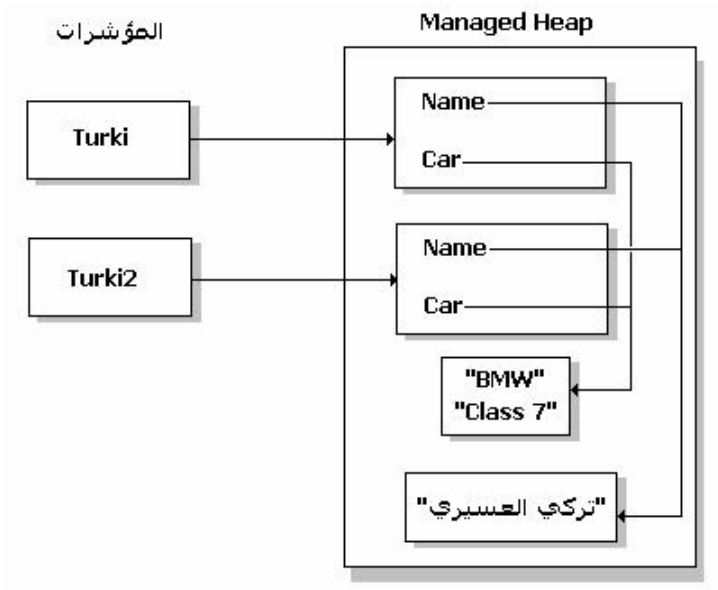
```
Dim Turki As New Person()
Dim Turki2 As Person

Turki.Name = "تركي العسيري"
Turki.Car.Name = "BMW"
Turki.Car.Model = "Class 7"

Turki2 = Turki.Clone

ArabicConsole.WriteLine(Turki2.Name)          ' تركي العسيري
ArabicConsole.WriteLine(Turki2.Car.Name)      ' BMW
```

من المخرجات السابقة، يتضح لنا ان الكائن المحضون Car قد تم نسخه بالفعل، وبذلك يكون مؤلف الكتاب مخطئا في عبارته "ان الكائنات الاخرى التي تحتويها الفئة لايمكن نسخها".

مع ذلك، مازلت مصراً على ان الكائن Car لم يتم نسخه بالفعل وانما **تم نسخ مؤشره الى الحقل الاخر** والتابع للكائن Turki2، والدليل سينفجر امام عينيك باستخدام المعامل Is الذي سيكشف لنا ان المؤشرين Turki.Car و Turki2.Car يشيران الى نفس نسخة الكائن، مما يعني أن الكائن لم يتم نسخه (**شكل 9-4**):

```
ArabicConsole.WriteLine(Turki.Car Is Turki2.Car) ' True
```

PDF



شكل 9-4: تم نسخ مؤشر الكائن المحضون Car فقط.

في عالم NET. يسمى النسخ السابق **بالنسخ السطحي Shallow Copying**، حيث أن النسخ لا يشمل الكائنات التابعة لخريطة الكائنات Object Graph، وانما مؤشراتها فقط. اما ان اردت نسخ جميع الكائنات الموجودة في خريطة الكائنات، فانك تريد تطبيق ما يسمى **بالنسخ العميق Deep Copying**. لعمل ذلك، لن تجد اسهل من استخدام التسلسل، اعد كتابة الطريقة Clone() في الفئة Person بهذا الشكل (ولا تنسى استخدام المواصفة Serializable Attributes في الفئات Person و Car):

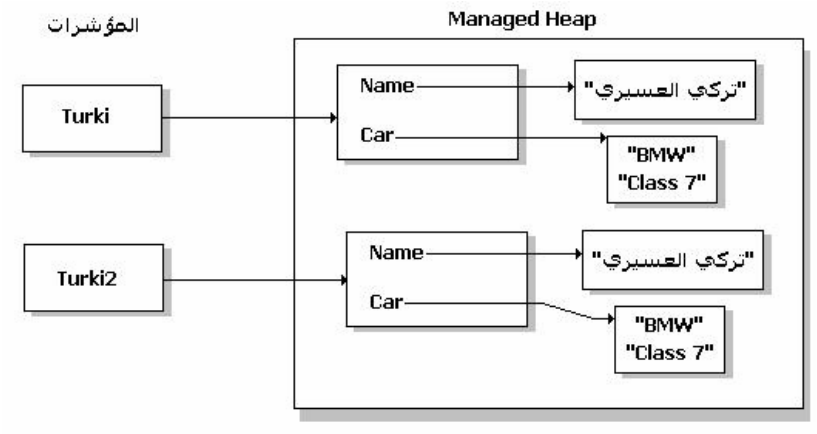
```
<Serializable()> _
Class Car
    …
    …
End Class

<Serializable()> _
Class Person
    Implements ICloneable
    …
    …
```

PDF

```
    Public Function Clone() As Person
        Dim st As FileStream = File.Open("C:\temp.tmp", _
            FileMode.OpenOrCreate)

        Dim SerialObj As New BinaryFormatter()

        SerialObj.Serialize(st, Me)
        st.Seek(0, SeekOrigin.Begin)
        Return CType(SerialObj.Deserialize(st), Person)
        st.Close()
        File.Delete("C:\temp.tmp")
    End Function
End Class
```

وهذا هو الاختبار الذي يثبت انه تم بالفعل نسخ جميع الكائنات المحضونة في الكائن:

```
Dim Turki As New Person()
Dim Turki2 As Person

Turki.Name = "تركي العسيري"
Turki.Car.Name = "BMW"
Turki.Car.Model = "Class 7"

Turki2 = Turki.Clone

ArabicConsole.WriteLine(Turki2.Name)          ' تركي العسيري
ArabicConsole.WriteLine(Turki2.Car.Name)      ' BMW

ArabicConsole.WriteLine(Turki.Car Is Turki2.Car) ' False ç اهم اثبات
```



شكل 9-5: النسخ العميق Deep Copying.

ملاحظة

كما تعلم فان الطريقة Serialize() التابعة للفئة BinaryFormatter تتطلب في وسيطتها الأولى كائن من النوع Stream لتطبيق التسلسل عليها. عندما اعدت كتابة الطريقة Clone() في الفئة Person الاخيرة، استخدمت وحدة التخزين FileStream لانشاء ملف مؤقت، يمكنك استخدام MemoryStream لزيادة السرعة.

انشاء Custom Serialization خاصة

قد تحتاج الى تطوير اسلوب خاص لتطبيق التسلسل على فئاتك الخاصة، استخدامك للمواصفة Serializable Attributes محدود جدا حيث أن كل ما ستفعله بهذا الاسلوب هو جعل جميع الحقول قابلة للتسلسل، صحيح انك تستطيع استخدام المواصفة NonSerialized Attributes لمنع التسلسل عن أي حقل، الا ان عملية المنع تتم وقت التصميم وليس التنفيذ. تخيل مثلا اننا نريد منع عملية التسلسل للحقول التي تكون قيمتها سالبه وقت التنفيذ، في هذه الحالة عليك تطوير تسلسلات خاصة Custom Serializations كما سترى في الفقرات التالية.

الواجهة ISerializable

ان اردت تطوير تسلسلات خاصة بك، فعليك تضمين الواجهة ISerializable في الفئة، تحتوي هذه الواجهة على طريقة واحدة فقط هي GetObjectData() وصيغتها:

```
Sub GetObjectData(ByVal info As SerializationInfo, _
    ByVal context As StreamingContext)
    …
    …
End Sub
```

يتم استدعاء هذه الطريقة تلقائيا بمجرد تطبيق التسلسل على الفئة، استخدم الوسيطة المرسلة الاولى info ان اردت اضافة بيانات تود تضمينها في تسلسل الكائن الحالي باستخدام طريقتها AddValue():

```
info.AddValue("Age", Me.Age)
```

عند تضمين الواجهة ISerializable في الفئة، لابد من انشاء مشيد مخفي بهذه الصيغة:

```
Private Sub New(ByVal info As SerializationInfo, _
    ByVal context As StreamingContext)
    …
    …
End Sub
```

سيتم استدعاء المشيد المخفي السابق لحظة عكس عملية التسلسل، يمكنك استخدام الوسيطة info ايضا لقراءة البيانات باستخدام الطرق GetXXX():

```
Me.Age = info.GetInt32("Age")
```

بما انك صرحت عن مشيد مخفي Private Constructer في الفئة، وكما ذكرت في الفصل الثالث **الفئات والكائنات** من انك لن تتمكن من انشاء نسخة كائن من هذه الفئة، لذلك عليك اعادة تعريف Overloads المشيد واضافة مشيد قابل للوصول حتى تتمكن من انشاء كائن من الفئة:

```
Public Sub New()

End Sub
```

PDF

مثال تطبيقي

في المثال التالي قمت بتطوير الفئة TestClass والتي تحتوي على اربعة حقول من النوع Integer (قد تفضل استخدام مصفوفة من النوع Integer ولكن هدفي هو توضيح الفكرة بشكل اسهل)، في هذه الفئة قمت بتضمين الواجهة ISerializable لتطوير تسلسل خاص بها، حيث ان الحقول موجبة القيمة هي التي سيتم تسلسلها فقط، اما السالبة فسيتم تجاهلها. ركز في شيفرات الاجراء GetObjectData() حيث يتم استدعائه لحظة التسلسل، والمشيد المخفي Private Sub New() لحظة عكس التسلسل:

```
<Serializable()> _
Class TestClass
    Implements ISerializable

    Public X As Integer
    Public Y As Integer
    Public Z As Integer
    Public W As Integer

    Public Sub New()
        ' لا احتاج هذا المشيد حاليا في هذه الفئة
        ' ولكن من الضروري تعريفه حتى تتمكن من
        ' انشاء كائن من هذه الفئة
    End Sub

   ' سيتم استدعاء هذا الاجراء لحظة عكس التسلسل
   ' Deserialization
    Private Sub New(ByVal info As SerializationInfo, _
               ByVal context As StreamingContext)

        On Error Resume Next
        Me.X = info.GetInt32("X")
        Me.Y = info.GetInt32("Y")
        Me.Z = info.GetInt32("Z")
        Me.W = info.GetInt32("W")

    End Sub

   ' سيتم استدعاء هذا الاجراء لحظة التسلسل
   Public Sub GetObjectData(ByVal info As SerializationInfo, _
               ByVal context As StreamingContext) Implements _
                              ISerializable.GetObjectData

        If Me.X >= 0 Then
            info.AddValue("X", Me.X)
        End If
        If Me.Y >= 0 Then
            info.AddValue("Y", Me.Y)
        End If
```

```
        If Me.Z >= 0 Then
            info.AddValue("Z", Me.Z)
        End If
        If Me.W >= 0 Then
            info.AddValue("W", Me.W)
        End If
    End Sub
End Class
```

وهنا مثال تطبيقي يقوم بتطبيق التسلسل/عكس التسلسل على كائن من الفئة السابقة (لاحظ ان الحقول السالبة لن يتم تسلسلها):

```
Dim st As FileStream = File.Open("C:\test.dat", FileMode.OpenOrCreate)
Dim SerialObj As New BinaryFormatter()
Dim TestObject As New TestClass()

TestObject.X = 1
TestObject.Y = -1
TestObject.Z = -5
TestObject.W = 5

SerialObj.Serialize(st, TestObject)

Dim tmp As TestClass
st.Seek(0, SeekOrigin.Begin)
tmp = CType(SerialObj.Deserialize(st), TestClass)

ArabicConsole.WriteLine(tmp.X) ' 1
ArabicConsole.WriteLine(tmp.Y) ' 0
ArabicConsole.WriteLine(tmp.Z) ' 0
ArabicConsole.WriteLine(tmp.W) ' 5
```

التسلسل بصيغة XML

ان كنت مقبلا على تطوير خدمات ويب Web Services، فقد تطبق التسلسل على الكائنات بصيغة XML القياسية، حيث يمكن تطبيق التسلسل على الكائنات بصيغة XML بسهولة شديدة وذلك بفضل الفئة XmlSerializer المقدمة من اطار عمل NET Framework..

قبل استخدام هذه الفئة من الضروري تنبيهك إلى ثلاث نقاط ضعف يمكن اعتبارها قصوراً كبيراً في الفئة:

الاولى: انه لا يمكنك تطبيق التسلسل الا على الفئات المعرفة بمحدد الوصول Public:

```
' يمكن تطبيق التسلسل بصيغة XML عليها
Public Class Person
    …
End Class

' غير ممكن
Class Car
    …
End Class
```

الثانية: ان الحقول العامة Public هي التي سيتم تطبيق التسلسل عليها فقط:

```
Public Class Test
    Public X As Integer        ' ممكن
    Friend Y As Integer        ' غير ممكن
    Private Z As Integer       ' غير ممكن
    …
End Class
```

اما النقطة الثالثة فتتلخص في فشل تطبيق التسلسل على خريطة الكائنات Object Graph فيما لو احتوت الخريطة على حالة المرجعية الدائرية Circular Reference بعكس الحال مع التسلسل بالصيغة الثنائية Binary.

مع كل هذا القصور في تطبيق التسلسل بالصيغة XML، قد تستغرب الاصرار على استخدامها؟ في اغلب الاحوال ستطبق التسلسل على كائناتك بصيغة XML لتبادل البيانات مع البرامج الاخرى خاصة عند تطوير خدمات ويب Web Services، وهي صيغة مفهومة لانها قياسية. الفقرات التالية ستكشف لك مزايا وامكانيات اضافية للتسلسل بصيغة XML.

ملاحظة

هذا القسم من الفصل يستخدم فئات اضافية عليك استيرادها من مجال الاسماء التالي لاختصار الشيفرات البرمجية:

Imports System.Xml.Serialization

الفئة XmlSerializer

ان كنت تنوي جعل فئاتك قابلة للتسلسل بصيغة XML، فلست مضطرا لاستخدام المواصفة Serializable Attribute، ولكنك بالتأكيد ستكون مضطراً لاستخدام محدد الوصول Public:

```
Public Class Person
    Public Name As String
    Public Age As Integer
End Class
```

ولتطبيق التسلسل بصيغة XML على الفئة السابقة، فاستخدم الفئة XmlSerializer (الموجودة ضمن مجال الاسماء System.XML.Serialization) واستخدامها يتم بنفس اسلوب استخدام الفئة BinaryFormatter ولكنها تتطلب تحديد نوع الكائن المراد تسلسله في مشيدها:

```
Dim st As FileStream = File.Open("C:\test.xml", FileMode.OpenOrCreate)
Dim SerialObj As New XmlSerializer(GetType(Person))
Dim Turki As New Person()

Turki.Name = "تركي العسيري"
Turki.Age = 99

SerialObj.Serialize(st, Turki)

st.Close()
```

الشيفرة السابقة ستقوم بانشاء الملف C:\Test.XML والذي سيحتوي بيانات الكائن بالصيغة XML:

```
<?xml version="1.0" ?>
    <Person xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
        <FullName>تركي العسيري</FullName>
        <Age>99</Age>
    </Person>
```

انظر ايضا

سأفترض في هذه الفقرة والفقرات التالية بانك تجيد التعامل مع لغة وصف البيانات XML، وان لم تكن كذلك فقد يفيدك الملحق أ: **لغة وصف البيانات** XML والملقى في نهاية هذا الكتاب.

PDF Converter

كما فعلنا سابقا مع الفئة BinaryFormatter، يمكنك عكس عملية التسلسل -ان كان بصيغة XML- باستدعاء الطريقة ()Deserialize والتابعة للكائن XmlSerializer، كما ترى في الشيفرة التالية:

```
Dim st As FileStream = File.Open("C:\test.xml", FileMode.OpenOrCreate)
Dim SerialObj As New XmlSerializer(GetType(Person))

st.Seek(0, SeekOrigin.Begin)
Dim Test As Person = CType(SerialObj.Deserialize(st), Person)

ArabicConsole.WriteLine(Test.Name)    ' تركي العسيري
ArabicConsole.WriteLine(Test.Age)     ' 99

st.Close()
```

مواصفات اضافية

يوفر لك اطار عمل NET. مجموعة كبيرة من المواصفات Attributes التي تعطيك تحكما وسيطرة اكبر على عملية التسلسل بصيغة XML. من هذه المواصفات:

المواصفة XmlRoot Attributes:

تمكنك المواصفة XmlRoot Attributes من تحديد الاسم للعنصر الجذري Root XML Element في مستند XML Document، بالاضافة الى مجال الاسماء Namespace للعنصر. يتم استخدام هذه المواصفة في الفئة:

```
<XmlRoot("PersonRecord", namespace:="http://www.dev4arabs.com")> _
Public Class Person
    Public Name As String
    Public Age As Integer
End Class
```

عند تطبيق التسلسل على كائن منشأ من الفئة السابقة، ستلاحظ ان اسم العنصر الجذري هو PersonRecord وليس اسم الفئة Person:

```
<?xml version="1.0" ?>
    <PersonRecord xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns="http://www.dev4arabs.com">
        <Name>تركي العسيري</Name>
        <Age>99</Age>
    </PersonRecord>
```

PDF Factory Pro trial version www.fastio.com

المواصفة XmlElement Attributes:

عند تطبيق التسلسل على كائن بصيغة XML، فسيتم حفظ اسماء حقوله في وسوم XML Tags كما هي موجودة بالشيفرة المصدرية. اما ان رغبت في تغيير اسم الحقل عند تسلسله بالصيغة XML، فاستخدم المواصفة XmlElement Attributes:

```
Public Class Person
    <XmlElement("FullName")> Public Name As String
    Public Age As Integer
End Class
```

عند تطبيق التسلسل على كائن منشأ من الفئة السابقة، ستلاحظ ان الحقل Name قد تمت تسميته بـ FullName في الوسم الخاص به:

```
<?xml version="1.0" ?>
    <Person xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
        <FullName>تركي العسيري</FullName>
        <Age>99</Age>
    </Person>
```

المواصفة XmlAttributeAttribute Attributes:

تمكنك المواصفة XmlAttributeAttribute Attributes من تسلسل الكائن وحفظه كواصف لمستند XML Documents بدلا من عنصر Element، فالحقل ID التالي:

```
Public Class Person
    …
    …
    <XmlAttributeAttribute("PersonID")> Public ID As String
    …
    …
End Class
```

ستكون مخرجاته مشابهة للتالي:

```
<?xml version="1.0" ?>
    <Person xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        PersonID="1">
        <Name>تركي العسيري</Name>
        <Age>99</Age>
    </Person>
```

المواصفة XmlText Attributes:

تستخدم المواصفة XmlText Attributes على الحقل ليتم تسلسله دون استخدام وسوم XML Tags:

```
Public Class Person
    …
   <XmlText()> Public Note As String
    …
End Class
```

عند تسلسل الحقل السابق، سيتم تسلسل قيمته فقط ويتم تجاهل وسوم XML Tags، بافتراض ان قيمة الحقل = "ملاحظات اضافية حول الشخص"، ستكون مخرجات الملف شبيهه بـ:

```
<?xml version="1.0" ?>
    <Person xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
        <Name>تركي العسيري</Name>
        <Age>99</Age>
        ملاحظات اضافية حول الشخص
    </Person>
```

المواصفتان XmlArray and XmlArrayItem Attributes:

تعمل المواصفتان XmlArray and XmlArrayItem Attributes جنبا الى جنب عند التعامل مع الحقول من نوع المصفوفات (وبالتحديد مصفوفات الكائنات)، حيث تستخدم المواصفة الاولى XmlArray Attributes لتعريف اسم عنصر XML Elemetent الخارجي بينما المواصفة الثانية للعنصر الداخلي. راقب الفئة Customer التالية:

```
Public Class Order
    <XmlAttributeAttribute("OrderId")> Public OrderID As String
    Public [Date] As Date
    Public Total As Decimal
End Class

Public Class Customer
    Public Name As String
    Public Address As String
    <XmlArray("CustomerOrders"), XmlArrayItem("CustOrder")> _
    Public Orders(3) As Order
End Class
```

قم بانشاء كائن من الفئة Customer السابقة، واسند قيم لجميع خصائصها وحاول تطبيق التسلسل على الكائن بصيغة XML في ملف خارجي، ليظهر لك شيئا مثل (لاحظ الوسوم بالخط السميك Bold لتعرف كيف تؤثر المواصفتان XmlArray and XmlArrayItem Attributes في المخرجات):

```
<?xml version="1.0" ?>
    <Customer xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
            <Name>تركي العسيري</Name>
            <Address>المملكة العربية السعودية</Address>
        <CustomerOrders>
            <CustOrder OrderId="1">
                <Date>2002-12-12T02:09:48.9743296+03:00</Date>
                <Total>100</Total>
            </CustOrder>
            <CustOrder OrderId="2">
                <Date>2002-12-10T00:00:00.0000000+03:00</Date>
                <Total>200</Total>
            </CustOrder>
            <CustOrder OrderId="3">
                <Date>2002-12-09T00:00:00.0000000+03:00</Date>
                <Total>300</Total>
            </CustOrder>
            <CustOrder xsi:nil="true" />
        </CustomerOrders>
    </Customer>
```

المواصفة XmlIgnore Attributes:

اخيرا، إرفاقك للمواصفة XmlIgnore Attributes قبل اسم الحقل سيتم تجاهله ولن تطبق عليه عملية التسلسل. فالحقل MotherName التالي لن يتم شمله في التسلسل:

```
Public Class Person
    …
    …
    <XmlIgnore()> Public MotherName As String
    …
    …
End Class
```

PDF Factory Pro trial version www.fastio.com

أحداث تقع عند عكس التسلسل

تحتوي الفئة XmlSerializer على اربعة احداث يتم اطلاقها عند عكس التسلسل على الكائنات وبالتحديد لحظة قراءة عنصر غير معرف في وحدة التخزين بالصيغة XML. وبما ان وحدة التخزين ستكون -في اغلب الاحوال- هي ملفات XML، فان امكانية تعديل هذه الملفات من برامج التحرير المختلفة امر قد يحدث ولو بطريق الخطأ. لذلك، قد تحتاج الى هذه الاحداث التابعة للكائنات التي تنشئها من هذه الفئة.

اعرض لك في هذه الفقرة طريقة استخدام الحدث UnknownElement والذي يتم إطلاقه بمجرد وجود عنصر غير معرف في وحدة التخزين بصيغة XML، يمكنك الاستفادة من هذا الحدث لمعرفة العناصر الغير معرفة والتي لا تتوافق مع عناصر فئة الكائن الذي يجري عكس تسلسله. ان لم تكن قد كتبت شيئا في هذا الحدث وتم العثور على عنصر غير معرف، فسيتم تجاهل هذا العنصر ولن يتم استرجاعه في حقله الافتراضي.

مع ذلك، قد تستفيد من هذا الحدث بتعبئة الحقول التي تم تسلسلها في ملفات XML يدويا ان كانت العناصر غير معرفة. مثلا، افترض انه تم تعريف الفئة Person التالية:

```
Public Class Person
    Public Name As String
    Public Age As Integer
End Class
```

عند تطبيق التسلسل على كائن منشأ من الفئة السابقة، ستحتوي وحدة التخزين بصيغة XML على شيئا مثل:

```
<?xml version="1.0" ?>
    <Person xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
        <Name>تركي العسيري</Name>
        <Age>99</Age>
    </Person>
```

والان تخيل ان احد المستخدمين قد احدث تغييراً في محتويات الملف السابق، وقام بتقسيم العنصر Name الى عنصرين هما FirstName و LastName:

```
<?xml version="1.0" ?>
    <Person xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
        <FirstName>تركي</FirstName>
        <LastName>العسيري</LastName>
```

PDF Editor

```
        <Age>99</Age>
    </Person>
```

عند عكس التسلسل على الملف السابق، سيتم استعادة الحقل Age بشكل صحيح، اما الحقل Name فلن يحتوي على شئ والسبب واضح. لذلك، عليك قراءة العنصرين في ملف XML ووضعهما في المكان المناسب، اكتب اجراء جديد باسم UnknowXMLElemntEvent مثلا واملأه بهذه الشيفرة:

```
Sub UnknowXMLElemntEvent(ByVal sender As Object, _
    ByVal e As XmlElementEventArgs)

    Dim tmpObj As Person = CType(e.ObjectBeingDeserialized, Person)

    If e.Element.Name = "FirstName" Then
        tmpObj.Name = e.Element.InnerXml
    ElseIf e.Element.Name = "LastName" Then
        tmpObj.Name = tmpObj.Name & " " & e.Element.InnerXml
    End If
End Sub
```

الاجراء السابق سيتم استدعائه في كل مرة يتم اكتشاف عنصر غير معرف لحظة عكس التسلسل، الخاصية e.ObjectBeingDeserialized تمثل الكائن الذي يتم عكس تسلسله، والخاصية e.Element.Name هي اسم العنصر في وحدة التخزين XML، اما الخاصية e.Element.InnerXml فتمثل القيمة نفسها. الاجراء السابق سيدمج العنصرين FirstName و LastName في حقل واحد وهو Name. يمكنك تجربة الملف والاجراء السابق بعكس عملية التسلسل، ولا تنسى قنص الحدث UnknownElement بالكلمة المحجوزة WithEvents او باستخدام الامر AddHandler كما فعلت في الشيفرة التالية:

```
Dim st As FileStream = File.Open("C:\test.xml", _
    FileMode.OpenOrCreate)
Dim SerialObj As New XmlSerializer(GetType(Person))


st.Seek(0, SeekOrigin.Begin)
' قنص الحدث اولا
AddHandler SerialObj.UnknownElement, AddressOf UnknowXMLElemntEvent
' عكس التسلسل
Dim Test As Person = CType(SerialObj.Deserialize(st), Person)

ArabicConsole.WriteLine(Test.Name)    ' تركي العسيري
ArabicConsole.WriteLine(Test.Age)     ' 99

st.Close()
```

هذا ليس كل شيء، ولا أريد منك أن تعتقد انك عرفت الكثير عن تسلسل الكائنات Object Serialization بعد قراءتك للصفحات السابقة، حيث ان هذا الفصل لا يعتبر الا مقدمة بسيطة الى التسلسل وطرق تطبيقه بـ Visual Basic .NET. ان كنت من المبرمجين الجادين و مهتما بهذا الموضوع، فأنصحك بقضاء الأسابيع القادمة في دراسة المزيد والمزيد من التفاصيل المتقدمة حول التسلسل من مكتبة MSDN او البحث عن مقالات اخرى في شبكة الانترنت. اما الصفحات التالية فستأخذك الى موضوع اخر –موجه ايضا للمبرمجين الجادين- وهو **مسارات التنفيذ Threading**.

PDF

مسارات التنفيذ Threading

ان كانت برمجة إجراءات Windows API تتطلب مبرمجين شجعان، فان برمجة مسارات التنفيذ المتعددة Multithreading تتطلب مبرمجين لا يخشون لومة لائم ولديهم استعداد للتضحية بعقولهم ومنطقهم البرمجي.

هذا الفصل مخصص بأعقد موضوع من مواضيع برمجة إطار عمل NET Framework. وهو مسارات التنفيذ Threading، وان كنت مبرمج لا ينوي التلاعب بمسارات التنفيذ في حياته البرمجية، فأنصحك بتجاهل هذا الفصل حتى تنعم بحياة برمجية اجمل!

ملاحظة

استرد مجال الأسماء التالي، حيث ان جميع فئات مسارات التنفيذ المذكورة في هذا الفصل مشمولة فيه:

Imports System.Threading

مقدمة إلى مسارات التنفيذ

مسار التنفيذ Thread عبارة عن مسار يتبعه المعالج لتنفيذ الشيفرة. يكون مسار التنفيذ Thread تابع لبرنامج عامل Process واحد فقط (البرنامج عندما يتم تنفيذه ويكون بالذاكرة يطلق عليه **برنامج عامل Process**). قد يحتوي البرنامج على أكثر من مسار تنفيذ، وفي هذه الحالة نطلق عليه برنامج **متعدد مسارات التنفيذ Multi-Threaded Process**. جميع البرامج التي طورناها من الفصل الأول لهذا الكتاب حتى هذا الفصل تسمى **برامج أحادية مسار التنفيذ Single Threaded Process** والسبب انها تحتوي على مسار تنفيذ واحد فقط (بغض النظر عن مسارات التنفيذ المخفية الناتجة من استخدام بعض كائنات إطار عمل NET Framework. كالمجموعة Garbage Collection مثلا).

لن أنتقل إلى التفاصيل الأخرى حتى أتأكد من فهمك واستيعابك لمسارات التنفيذ، وقد يقرب لك هذا المثال مفهوم مسارات التنفيذ بصورة افضل، نفترض ان لديك برنامج يقوم بثلاث عمليات في وقت واحد وهي التدقيق الإملائي للبيانات، حفظ وتخزين البيانات، اظهار البيانات على الشاشة. لاحظ أنني ذكرت ثلاثة عمليات 1) تدقيق إملائي 2) تخزين 3) إظهار، وبما ان هذه العمليات الثلاثة تتم في وقت واحد، نستطيع ان نقول ان هذا البرنامج يحتوي على ثلاثة مسارات تنفيذية Three Threads. فهنا يوجد مسار تنفيذ خاص لإجراء عملية التدقيق الإملائي، و مسار تنفيذ خاص للتخزين، وأخيرا مسار تنفيذ خاص لإظهار البيانات. اما لو كان البرنامج يحتوي على مسار تنفيذ واحد، فلابد ان تتم كل عملية بعد نهاية الأخرى -أي لا تستطيع إجراءها في وقت واحد. لذلك، سيقوم البرنامج بعملية التدقيق الإملائي ثم عملية التخزين أو الحفظ وأخيرا إظهار البيانات على الشاشة.

قد تفيدك مسارات التنفيذ المتعددة في تنفيذ عدة عمليات في وقت واحد، كأن تجعل البرنامج يقوم بعملية الطباعة مثلا أو الاستعلام في قاعدة بيانات، بينما يقوم نفس البرنامج بعرض بيانات أخرى على الشاشة أو إعطاء فرصة للمستخدم بإلغاء العملية أو حتى عمل أي شيء آخر. وحتى اذا قمت بتطوير برنامج يعمل في جهاز خادم Server لشبكة محلية مثلا، فأنت بالفعل تحتاج إلى مسارات التنفيذ المتعددة حتى تمكن جميع المستخدمين لهذا الخادم من استخدامه دون الانتظار في طابور، فلو كان برنامجك لا يوجد به إلا مسار تنفيذ واحد فقط، فلن يستطيع إلا ان يخدم عميل Client واحد في وقت واحد، مما يضطر أي مستخدم انتهاء العملية الحالية التي يقوم بها البرنامج لخدمة مستخدم آخر ومن ثم خدمته. وهو بلا شك، شيء مرفوض في هذا العصر من تطوير التطبيقات.

مع ذلك، التلاعب بمسارات التنفيذ في شيفراتك المصدرية فيه شيء كبير من الخطر والحصول على نتائج سلبية غير متوقعة. ليس هذا فقط، بل ان إدارة مسارات التنفيذ في برامجك عملية معقدة وذلك بسبب ان التنفيذ لا يتبع مسار منطقي واحد، وانما مجموعة من المسارات التي تجعلك تعيد النظر عشرات المرات قبل اعتماد برنامجك. حيث ان نجاح تنفيذ البرنامج في المرة الأولى، لا يشترط تنفيذه بنجاح في التجربة الثانية أو الثالثة أو حتى العاشرة!

أنواع مسارات التنفيذ

بصفة عامة، يوجد نوعان من مسارات التنفيذ: **مسارات التنفيذ الحرة Free Threading** و**مسارات التنفيذ المتباعدة Apartment Threading**. في مسارات التنفيذ الحرة، يكون كل مسار تنفيذ قابل للوصول إلى جميع بيانات البرنامج والمتغيرات العامة Global Variable التي

فيه. فلو كان البرنامج يحتوي على مسارين تنفيذين Two Threads، فكلا هذين المسارين يستطيعان الوصول إلى جميع المتغيرات العامة للبرنامج.

رغم ان هذا النوع من مسارات التنفيذ يعطيك مرونة كبيرة، إلا ان التعامل معه يتطلب مهارة كبيرة، وكثرة الشوائب والأخطاء التي به صعبة الاستكشاف، وذلك بسبب التعارضات التي قد تحدثها مسارات التنفيذ المختلفة على المتغيرات العامة في البرنامج. ولإعطائك فكرة عن مثل هذه التعارضات، افترض –مثلا- ان لدينا برنامج يحتوي على متغير عام باسم X، ولهذا البرنامج مسارين تنفيذين هما A و B، ويحتوي البرنامج في احد سطوره على هذه الشيفرة:

```
10 If X < 0 Then
20     ArabicConsole.WriteLine ("قيمة المتغير X اقل من الصفر")
30 End If
```

والان راقب هذا السيناريو، نفترض ان مسار التنفيذ A جعل قيمة المتغير X اقل من صفر، وعندما وصل مسار التنفيذ إلى السطر 10 سيحقق الشرط -لان قيمة X اصغر من صفر، وقبل ان ينتقل مسار التنفيذ إلى السطر 20 قام مسار التنفيذ الثاني (وهو B) بزيادة قيمة المتغير X إلى قيمة اكبر من صفر، فعندما يصل مسار التنفيذ A إلى السطر 20، سينفذ الشيفرة وسيظهر الرسالة والخاصة بقيمة المتغير X وهي بلا شك رسالة خاطئة المنطق والتوضيح.

اما مسارات التنفيذ المتباعدة Apartment Threading ففيها يتم تخصيص المتغيرات العامة في البرنامج لكل مسار التنفيذ. فالمتغير العام X -في المثال السابق- ستكون له نسختان، نسخة لمسار التنفيذ A ونسخة أخرى لمسار التنفيذ B، مما يضمن عدم حدوث التعارضات في المتغيرات العامة لمسارات التنفيذ المتعددة. في Visual Basic .NET يمكنك من إنشاء مسارات تنفيذ حرة، ولكنك ستحتاج إلى **التزامن Synchronization** حتى لا تسبب التعارض بين المتغيرات العامة –كما سترى لاحقا في هذا الفصل.

متى تستخدم مسارات التنفيذ المتعددة؟

معظم المبرمجين يعتقدون ان مسارات التنفيذ المتعددة تزيد من عملية سرعة المعالجة، وذلك بسبب قابليتها لتنفيذ المهام المتعددة في وقت واحد. قد يكون هذا الكلام صحيح في حالات معينة وخاصة في الاجهزة التي تحتوي على أكثر من معالج Processor، لأن كل معالج سيقوم بتنفيذ تعليمات مسار التنفيذ بشكل مستقل عن الآخر. لكن في معظم الاحوال، سيحتوي الجهاز على معالج واحد

PDF Factory Pro trial version www.fastio.com

فقط، أي ان عملية تنفيذ جميع مسارات التنفيذ يقوم بها معالج واحد فقط، وبالتالي لن تكون هنالك سرعة اضافية.

أرجو ان لا تعتقد ان مسارات التنفيذ المتعددة قد تسبب في إبطاء سرعة المعالجة، لانها في الحقيقة لا تزيدها ولا تنقصها، لكن القضية تعتمد على نوعية وكثرة المهام. وسنأخذ هاتين الحالتين اللتان تبينان لك عيوب ومزايا مسارات التنفيذ المتعددة:

الحالة الأولى: نفترض ان البرنامج سيقوم بتنفيذ مهمتين، الواحدة منها تستغرق 10 ثواني، فلو كان مسار التنفيذ احادي Single Thread سيقوم بتنفيذ المهمة الأولى التي تستغرق 10 ثواني ومن ثم المهمة الثانية التي تستغرق 10 ثواني ويصبح المجموع للمهمتين 20 ثانية وسيكون المتوسط (20 + 10) ÷ 2 = 15 ثانية لكل مهمة. اما لو كانت مسارات التنفيذ متعددة، فان المهمتان سيتم انجازهما بشكل متوازي، ويكون المعدل دائما 10 + 10 = 20 ثانية لكل مهمة (طبعا لو وجد معالجين في الجهاز سيكون المعدل 10)، مما يبين لنا ان مسارات التنفيذ الأحادية افضل من مسارات التنفيذ المتعددة.

الحالة الثانية: نفترض ان لدينا مهمتين، الأولى تستغرق 10 ثواني والثانية تستغرق ثانية واحدة. في مسارات التنفيذ الأحادية، سيتم عملية تنفيذ المهمة الواحدة في وقت يتراوح ما بين 1 ثانية إلى 11 ثانية، مما ينتج عنه معدل مقارب إلى (11 + 1) ÷ 2 = 6 ثانية لكل مهمة. اما مع مسارات التنفيذ المتعددة، فان المهمة الثانية ستستغرق وقت يتراوح ما بين 1 ثانية إلى 2 ثانية مما يؤدي إلى معدل مقارب إلى 1.5 ثانية للمهمة الثانية. وبالتالي يتضح إلى ان مسارات التنفيذ المتعددة افضل من مسارات التنفيذ الأحادية.

إذاً، ننهي هذه الفقرة ونقول: ان مسارات التنفيذ المتعددة ليست افضل من مسارات التنفيذ الاحادية والعكس صحيح. عليك ان تحدد النوع المناسب في الحالة المناسبة ونوع الحاجة. أستطيع ان اقول ان مسارات التنفيذ المتعددة هي افضل في حال كون الوقت المستغرق لإنجاز المهام مختلف من مهمة إلى أخرى. وقد تحتاج إلى مسارات التنفيذ المتعددة اذا كنت تود من القيام بعملية تنفيذ مهمات في الخلفية Background دون ان توقف عمل المستخدم. فلو تلاحظ طريقة عمل مدقق الإملاء الفوري Auto Spell-Checker التابع لبرنامج Microsoft Word، ستكتشف انه يعمل في مسار تنفيذ خاص ومستقل مما يجعلك قادرا على الكتابة في أي وقت تريده بينما يقوم المدقق بالتدقيق الإملائي على المستند في الخلفية بنفس الوقت.

ملاحظة

حكمة برمجية ضعها دائما في عين اعتبارك: كلما زاد عدد مسارات التنفيذ في شيفراتك، كلما زاد تعقيدها وصعوبة اكتشاف أخطائها.

إنشاء مسار تنفيذ

في إطار عمل NET Framework.، مسار التنفيذ ما هو إلا كائن منشأ من الفئة Thread يتطلب مشيده إجراء مفوض Delegate باستخدام الكلمة المحجوزة AddressOf (لابد ان لا يحتوي الإجراء على أية وسيطات Parameters)، راقب هذا المثال:

```
Sub Main()
    Dim counter As Integer
    Dim newThread As New Thread(AddressOf DoSomething)

    newThread.Start()

    For counter = 1 To 1000
        Console.Write("$"c)
    Next

End Sub

Sub DoSomething()
    Dim counter As Integer

    For counter = 1 To 1000
        Console.Write("*"c)
    Next
End Sub
```

مخرجات الشيفرة السابقة، ستكون شيئا شبيها بـــ:

```
$$$
```

في الشيفرة السابقة، قمت بإنشاء مسار تنفيذ جديد وأسندته إلى المؤشر newThread، وبعد ذلك قمت باستدعاء الطريقة Start() ليتم تشغيل مسار التنفيذ وتنفيذ الإجراء المفوض له (وهو DoSomething())، من الضروري ان أذكرك هنا بان السطور التي تلي الاستدعاء newThread.Start() سيتم تنفيذها أيضا في نفس الوقت.

الطرق والخصائص

إن كانت الطريقة Start() تشغل مسار التنفيذ، فان الطريقة Abort() توقف عمله. يمكنك استدعاء الطريقة من خلال الخاصية Thread.CurrentThread والتي تعود بكائن من النوع Thread يمثل مسار التنفيذ الحالي:

```
…
…
Sub DoSomething()
    Dim counter As Integer

    For counter = 1 To 1000
        If counter = 100 Then
            Thread.CurrentThread.Abort() ' ايقاف مسار التنفيذ الحالي
        End If
        Console.Write("*"c)
    Next
End Sub
```

الطريقة Abort() توقف عمل مسار التنفيذ وتنفيه من الحياة بشكل نهائي، ولكن توجد طريقة أخرى توقف عمل مسار التنفيذ بشكل مؤقت ولا تنهيه وهي Suspend()، والتي قد تحتاج إتباعها بالطريقة Resume() لتكمل عمل مسار التنفيذ:

```
Dim newThread As New Thread(AddressOf DoSomething)
…
…

' تشغيل مسار التنفيذ
newThread.Start()
…
…

' ايقافه بشكل مؤقت
newThread.Suspend()
…
…
```

```
' متابعة التنفيذ (بعد الايقاف المؤقت)
newThread.Resume()
…
…

' انهاء التنفيذ بشكل نهائي
newThread.Abort()
…
…
```

المزيد أيضا، الطريقة Sleep() تمكنك من إيقاف عمل مسار التنفيذ (بشكل مؤقت) لفترة زمنية وحدتها 0.001 ثانية:

```
Thread.CurrentThread.Sleep(500) ' ايقاف مسار التنفيذ الحالي لمدة نصف ثانية
```

من الطرق أيضا الطريقة Join()، والتي توقف عمل مسار التنفيذ الحالي بشكل مؤقت حتى ينتهي تنفيذ مسار التنفيذ الآخر:

```
Dim newThread As New Thread(AddressOf DoSomething)
…
…

' تشغيل مسار التنفيذ
newThread.Start()

' سيتم تنفيذ هذه الشيفرة أيضا
…
…

newThread.Join ()
' لن يتم تنفيذ هذه الشيفرة حتى
' يموت مسار التنفيذ الثاني
…
…
```

اما الحديث عن الخصائص، فتستطيع معرفة ما اذا كان مسار التنفيذ قيد التنفيذ بالاستعلام عن الخاصية IsAlive، ليمكنك –مثلا- تطبيقها في الشيفرة السابقة عوضا عن الطريقة Join():

```
Dim newThread As New Thread(AddressOf DoSomething)
…
…

' تشغيل مسار التنفيذ
newThread.Start()
```

```
' سيتم تنفيذ هذه الشيفرة أيضا
…
…

Do While newThread.IsAlive() = True

Loop
' لن يتم تنفيذ هذه الشيفرة حتى
' يموت مسار التنفيذ الثاني
…
…
```

ملاحظة

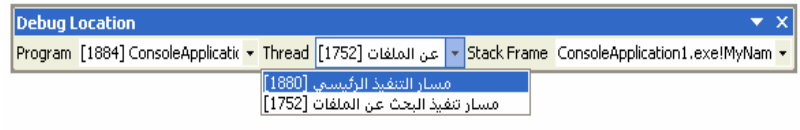
من الغباء الاستعلام عن الخاصية IsAlive لمسار التنفيذ الحالي:

```
If Thread.CurrentThread.IsAlive Then ...
```

فهي ستعود بالقيمة True دائما لأسباب منطقية.

الخاصية Name خاصية حرفية من النوع String قابلة للقراءة والكتابة، وفي الحقيقة ليس لها أي تأثير عملي على مسار التنفيذ لحظة التنفيذ، ولكنها تفيدك كثيرا لحظة التنقيح Debugging، حيث يظهر اسم مسار التنفيذ بتلك الخاصية في شريط الأدوات Debug location (**شكل 10-1**):

```
…
…
Thread.CurrentThread.Name = "مسار التنفيذ الرئيسي"
newThread.Name = "مسار تنفيذ البحث عن الملفات"
```



شكل 10-1: ظهور أسماء مسارات التنفيذ في شريط الأدوات Debug Location.

المزيد أيضا، تمكنك الخاصية ThreadState من معرفة حالة مسار التنفيذ، فالشرط التالي يختبر حالة مسار التنفيذ ما ان تم تشغيله أو لا:

```
If newThread.ThreadState = ThreadState.Unstarted Then …
```

بقية القيم للحالات الأخرى التي يكون عليها مسار التنفيذ يعرضها الجدول التالي:

القيمة	الحالة
ThreadState.Aborted	انهي مسار التنفيذ بشكل نهائي.
ThreadState.AbortRequested	تم طلب ايقاف لمسار التنفيذ.
ThreadState.BackGround	مسار تنفيذ خلفي BackGround Thread.
ThreadState.Running	يعمل.
ThreadState.Suspended	الإيقاف المؤقت.
ThreadState.SuspendedRequested	تم طلب إيقاف مؤقت لمسار تنفيذ.
ThreadState.UnStarted	لم يبدأ بعد.
ThreadState.WaitSleepJoin	تم استدعاء الطريقة Join على مسار التنفيذ.

خاصية الأسبقية Priority:

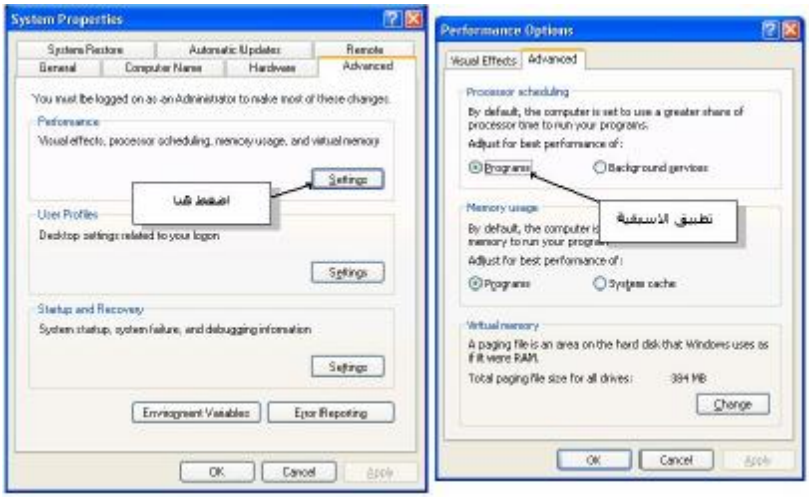
أنت تعلم وانا اعلم ان نظام التشغيل Windows هو نظام تشغيل **متعدد المهام Multitasking**، بحيث يمكنك من تشغيل أكثر من برنامج في وقت واحد، الامر الذي يفرض على النظام توزيع المعالجة بين التطبيقات المختلفة. في الإصدارات السابقة من نظام التشغيل Windows (الاصدار Windows ME وما قبله)، كانت تتم عملية توزيع **وقت المعالجة Processing Time** بشكل متوازي ومتساوي لكافة البرامج العاملة Processes، ووقت المعالجة ما هي إلا كمية الأوامر والتعليمات التي يقوم المعالج بتنفيذها في فترة زمنية معينة لبرنامج واحد. فلو وجدت 10 برامج عاملة في الذاكرة، سيقوم المعالج بتنفيذ جزء معين من كل برنامج لفترة متساوية.

التوزيع السابق لوقت المعالجة عادل لدرجة كبيرة، ولكن صفة العدل التي من شمائله لم تعد محبذة، وذلك بسبب وجود عشرات –ان لم يكن مئات- البرامج العاملة في وقت واحد (قد تكون

معظمها غير نشطة أو غير مستخدمة)، الأمر الذي يسبب بطء كبير في عملية تنفيذ **البرنامج النشط Active Process** (البرنامج الحالي).

لذلك، كان الحل الذي اعتمدوه مطورو نظام التشغيل Windows هو بإنتاج فلسفة **الأسبقية Priority**، والتي ظهرت مع الإصدارات Windows NT والتي تليها بحيث تعطي وقت معالجة أكثر للبرنامج النشط. لذلك، حتى لو وجدت مئات البرامج تعمل في الخلفية، فلن يعطيها نظام التشغيل حقها الكافي من وقت المعالجة، الامر الذي يترتب عليه نتائج ايجابية جدا للبرنامج الحالي مما يزيد من كفاءة تنفيذه.

تطبق الإصدارات الجديدة من نظم التشغيل Windows فلسفة الأسبقية بشكل افتراضي، ويمكنك التحقق من ذلك بنفسك ان تم اختيار الامر Programs في القسم Processing Scheduling في صندوق الحوار Performance Option، والذي تصل اليه عن طريق Control Panel -> System -> Advanced -> Performance -> Settings (**شكل 10-2**).



شكل 10-2: نظام التشغيل يعطي وقت معالجة أكثر للبرامج النشطة.

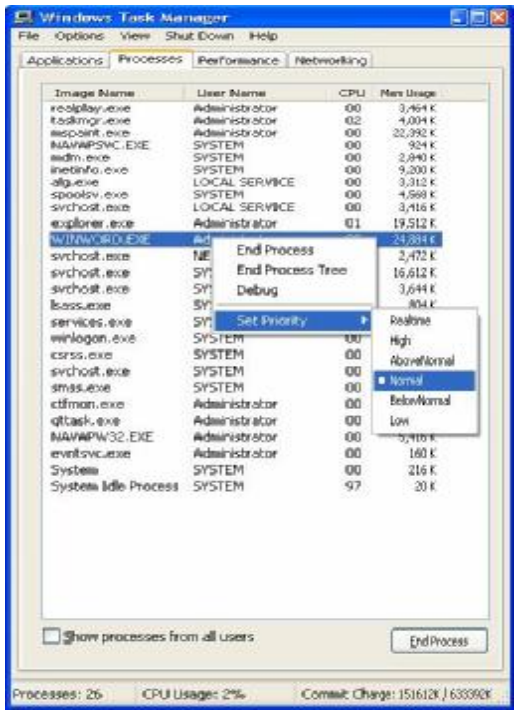
يمكنك نظام التشغيل من زيادة أو إنقاص وقت المعالجة لكل برنامج عامل، توجد ست مستويات لمقدار وقت المعالجة، هي -ابتداء من الأقل حتى الأكثر: Low، BelowNormal، Normal، AboveNormal، High، و Realtime. يمكنك التحكم فيها لكل برنامج عامل عن طريق صندوق الحوار Windows Task Manager والذي تصل اليه بالضغط على المفاتيح [Alt]+[Ctrl]+[Del] **(شكل 10- 3 في أعلى الصفحة التالية)**.

ملاحظة

تغير مقدار وقت المعالجة للبرامج العاملة يدويا بنفسك، قد يؤدي إلى نتائج سلبية غير متوقعة.

بعد هذه الفلسفة الطويلة حول الأسبقية، أردت ان أبين لك ان مسارات التنفيذ Threads في البرنامج العامل الواحد Process يمكن ان يكون وقت توزيع المعالجة لها مختلف من مسار تنفيذ إلى آخر، بل وتستطيع أيضا التحكم بمقدار وقت المعالجة عن طريق الخاصية Priority، والتي يمكنك إسناد قيمة لها من خمس قيم هي: ThreadPriority.BelowNormal، ThreadPriority.Lowest، ThreadPriority.Normal، ThreadPriority.AboveNormal، أو ThreadPriority.Highest.

PDF Factory Pro trial version www.fastio.com



شكل 10-3: تغيير مقدار وقت المعالجة لبرنامج عامل.

دعني اريك هاتين الشيفرتين والتي قد توضح لك مدى تأثير توزيع وقت المعالجة بين مسارات التنفيذ، الشيفرة الأول تكون فيها أسبقية كلا مسارات التنفيذ متساوية (بشكل افتراضي قيمة الخاصية Priority هي ThreadPriority.Normal لكل مسار تنفيذ):

```
Sub Main()
    Dim counter As Long
    Dim newThread As New Thread(AddressOf DoSomething)
    newThread.Start()

    ' قد تحتاج إلى زيادة أو نقصان مقدار عداد الحلقة
    ' على حسب سرعة المعالج عندك
    For counter = 1 To 200000000
    Next

    Console.WriteLine("THREAD A HAS FINISHED")

End Sub
```

```
Sub DoSomething()
    Dim counter As Integer

    For counter = 1 To 100
    Next
    Console.WriteLine("THREAD B HAS FINISHED")
End Sub
```

بما ان وقت المعالجة متساوي لكلا المسارين، فانه من البديهي سينتهي تنفيذ حلقة مسار التنفيذ الثاني قبل الأول، وذلك لان عداد الحلقة التكرارية في مسار التنفيذ الثاني لا يتجاوز 100، لتكون المخرجات بهذا الشكل:

```
THREAD B HAS FINISHED
THREAD A HAS FINISHED
```

لنحاول تغيير أسبقية كلا مسارات التنفيذ، ونزيد –مثلا- وقت المعالجة لمسار التنفيذ الأول وننقصها من الثاني:

```
Sub Main()
    Dim counter As Long
    Dim newThread As New Thread(AddressOf DoSomething)

    ' تعديل الاسبقية
    Thread.CurrentThread.Priority = ThreadPriority.Highest
    newThread.Priority = ThreadPriority.Lowest

    newThread.Start()

    ' قد تحتاج إلى زيادة مقدار عداد الحلقة
    ' على حسب سرعة المعالج عندك
    For counter = 1 To 200000000
    Next

    Console.WriteLine("THREAD A HAS FINISHED")

End Sub

Sub DoSomething()
    Dim counter As Integer

    For counter = 1 To 100
    Next
    Console.WriteLine("THREAD B HAS FINISHED")
End Sub
```

PDF

عند تنفيذك للشيفرة السابقة، ستفاجئ إن اكتشفت أن مسار التنفيذ الأول قد أنهى حلقته التكرارية (والتي عدادها 200 مليون مرة) قبل حلقة مسار التنفيذ الثاني، وذلك لأن وقت المعالجة المعطى لمسار التنفيذ الأول ThreadPriority.Highest أكثر بكثير من وقت المعالجة المعطى لمسار التنفيذ الثاني ThreadPriority.Lowest، لتكون المخرجات بهذا الشكل:

```
THREAD A HAS FINISHED
THREAD B HAS FINISHED
```

ملاحظة

نتائج الاختبارين الأخيرين استخلصتها من تنفيذ الشيفرة على جهازي الشخصي ذو معالج قديم من النوع Pentium 800 MHz، ولا اضمن لك نتائج متطابقة مع جهازك الشخصي.

خاصية الخلفية IsBackground:

سيستمر برنامجك في العمل ما دامت مسارات تنفيذه على قيد الحياة، بغض النظر عن **مسارات التنفيذ الخلفية Background Threads** حيث سيتم إنهائها بشكل تلقائي لحظة موت جميع مسارات التنفيذ العادية.

يمكنك معرفة ما اذا كان مسار التنفيذ خلفي Background Thread أو جعله خلفي عن طريق الخاصية IsBackground (وهي قابلة للقراءة والكتابة):

```
Dim newThread As New Thread(AddressOf DoSomething)
…
…
newThread.IsBackground = True
…
…
If newThread.IsBackground Then
    newThread.Abort()
End If
```

مسارات التنفيذ الخلفية هي نفس مسارات التنفيذ العادية، ويكمن الفرق الوحيد بينهما ان مسارات التنفيذ الخلفية لا تبقي البرنامج على قيد الحياة ان ماتت جميع مسارات التنفيذ الرئيسية في البرنامج. فمثلا، الشيفرة التالية ستبقي البرنامج على قيد الحياة (وذلك بسبب وجود مسار التنفيذ الثاني):

```
Module Module1
    Sub main()
        Dim newThread As New Thread(AddressOf DoSomething)

        newThread.Start()
    End Sub

    Sub DoSomething()
        Dim counter As Integer

        ' سيتم تنفيذ كامل الحلقة
        For counter = 1 To 1000
            Console.WriteLine(counter)
        Next

        Console.Read()
    End Sub
End Module
```

اما ان كان مسار التنفيذ الثاني من النوع الخلفي Background Thread، فسيتم إنهائه بمجـــرد انتهاء تنفيذ مسار التنفيذ الأول:

```
Module Module1
    Sub main()
        Dim newThread As New Thread(AddressOf DoSomething)

        ' جعله مسار تنفيذ خلفي
        newThread.IsBackground = True
        newThread.Start()
    End Sub

    Sub DoSomething()
        Dim counter As Integer

        ' لن يتم تنفيذ كامل الحلقة
        For counter = 1 To 1000
            Console.WriteLine(counter)
        Next

        Console.Read()
    End Sub
End Module
```

PDF

ملاحظة

لا تحاول استخدام الكائن ArabicConsole عند تجربة الشيفرة السابقة، وذلك لان البرنامج لن ينتهي بسبب ان ArabicConsole يحتوي على مسار تنفيذ خاص به يبقي البرنامج على قيد الحياة.

التعامل مع مسارات التنفيذ

بادئ ذي بدء عليك معرفة ان مسارات التنفيذ تعمل كالبرامج المستقلة، فيمكنك اعتبار ان كل مسار تنفيذ هو برنامج عامل، لدرجة انه لو حدث استثناء Exception في مسار تنفيذ، فلن تتأثر مسارات التنفيذ الأخرى وستكمل عملها كان شيئا لم يحدث:

```
Module Module1
    Sub main()
        Dim newThread As New Thread(AddressOf DoSomething)

        newThread.Start()
        …
        …
        ' سيتم استكمال تنفيذ الشيفرة التالية
        ' رغم وجود استثناء في مسار التنفيذ الثاني
        …
        …
    End Sub

    Sub DoSomething()
        Dim counter As Integer

        Throw New Exception()
        …
        …
    End Sub
End Module
```

رغم ان مسار التنفيذ الثاني قد رمى استثناء Throw Exception، إلا ان مسار التنفيذ الرئيسي سيكمل عمله بشكل طبيعي. مع ذلك، أنصحك بشدة بتفادي الاستثناء Catch Exception في مسار التنفيذ الثاني أو إيقاف عمل مسار التنفيذ الرئيسي حتى لا تنمو الشوائب والأخطاء في عمل البرنامج.

ملاحظة

لا تحاول تدارك الاستثناءات بين مسارات التنفيذ المختلفة فذلك لن يفيدك، فلو حاولت كتابة شيئا مثل:

```
Dim newThread As New Thread(AddressOf DoSomething)

Try
   newThread.Start()
Catch
   ...
End Try
```

فانك تتعب نفسك دون جدوى، حيث وجود التركيب Try ... End Try كعدمه ان كان الاستثناء قد رمي من مسار تنفيذ آخر.

ليس هذا فقط، بل حتى الاعدادات الإقليمية الحالية Regional Settings تختلف من مسار تنفيذ إلى آخر، والدليل ان كل مسار تنفيذ يحتوي على الخاصية CurrentCulture وهي خاصية من النوع CultureInfo تمكنك من تحديد الاعدادات الإقليمية لمسار التنفيذ الحالي:

```
Imports System.Globalization
…
…
Module Module1
    Sub main()
        Dim newThread As New Thread(AddressOf DoSomething)

        Thread.CurrentThread.CurrentCulture = New _
            CultureInfo("ar-SA")
        newThread.CurrentCulture = New _
            Globalization.CultureInfo("ar-EG")
        newThread.Start()

        MsgBox(100.ToString("C")) ' ر.س 100.00
    End Sub

    Sub DoSomething()
        MsgBox(100.ToString("C")) ' ج م 100.000
    End Sub
End Module
```

PDF Factory Pro trial version www.fastio.com

انظر أيضا

موضوع الاعدادات الإقليمية حديث ذو شجون لمحت إليه سابقا في الفصل السادس **الفئات الأساسية**.

على صعيد آخر، عمليات الإيقاف النهائي والإيقاف المؤقت باستخدام الطرق Abort() و Suspend() لا تتم فورا لحظة استدعائها، وانما حتى يصل مسار التنفيذ إلى **نقطة آمنة Safe Point**، والنقاط الآمنة Safe Points -استنادا إلى مراجع .NET Documentation- هي لحظة من الوقت يمكن للمجموعة Garbage Collection ان تبدأ عملها. مثلا، عندما ينهي إجراء معين تنفيذه ويعود بقيمة (راجع فهرس مستندات .NET Documentation. تحت العنوان **Safe Points** لمزيد من التفاصيل حول النقاط الآمنة).

ملاحظة

كانت أمنيتي عرض مثالا يوقف مسار تنفيذ وهو ليس في لحظة نقطة آمنة. ولكني -والاعتراف بالحق فضيلة – لست متأكدا حتى هذه لحظة من مدى استيعابي للنقاط الامنة. لذلك، دعني اكرر كلمتي الاخيرة وأنصحك بالانتقال إلى مكتبة MSDN لعل وعسى تطلع منها بشيء.

اما الحديث عن الأسبقية، فأنصحك بشدة عدم المحاولة بتعديلها وتغييرها بنفسك إلا ان دعت حاجة ماسة لذلك. بصفة عامة، يفضل إعطاء وقت معالجة اكبر لمسارات التنفيذ التي تتعامل مع المخرجات الظاهرية على الشاشة، حتى تتمكن من تنفيذ شيفرات العرض على المستخدم بأسرع ما يمكن.

متى يموت مسار التنفيذ؟

يموت مسار التنفيذ في حالة من ثلاث حالات: الأولى عندما يتم إيقافه باستخدام الطريقة Abort()، الثانية عندما ينتهي عمل الإجراء المفوض (الذي يعتبر نقطة البداية لمسار التنفيذ)، والثالث عند تنفيذ العبارة Exit Sub في الإجراء المفوض.

الحالات الثلاث السابقة تطبق أيضا على مسارات التنفيذ الخلفية Background Threads ولكن توجد حالة رابعة إضافية لها، وهي –كما أخبرتك- انها تموت إن ماتت جميع مسارات التنفيذ العادية في البرنامج بشكل تلقائي.

حالة غريبة حدثت لي عندما لعبت قليلا مع مسارات التنفيذ لم أجد تفسيرا لها حتى هذه اللحظة، وهي استدعاء الطريقة Abort() من أول مسار تنفيذ للبرنامج (مسار التنفيذ الرئيسي)، فالشيفرة التالية:

```
Module Module1
    Sub main()
        Dim newThread As New Thread(AddressOf DoSomething)

        newThread.Start()

        Thread.CurrentThread.Abort()
    End Sub

    Sub DoSomething()
        …
        …
    End Sub
End Module
```

ستنهي تنفيذ البرنامج بأكمله رغم وجود مسار تنفيذ آخر على قيد الحياة، فعندما جعلت مسار التنفيذ الرئيسي يموت دون استخدام Abort():

```
Module Module1
    Sub main()
        Dim newThread As New Thread(AddressOf DoSomething)


        Thread.CurrentThread.Abort()
    End Sub

    Sub DoSomething()
        …
        …
    End Sub
End Module
```

تم إكمال عمل مسار التنفيذ الثاني دون أي مشاكل. ولكن بعد عبث في أدوات التنقيح التابعة لبيئة Visual Studio .NET اكتشفت ان مسار التنفيذ الرئيسي لم يمت رغم عدم وجود أي شيفرة مصدرية قيد التنفيذ!

أستطيع ان استنتج من هذه التجربة الحالة الخامسة التي تموت فيها مسارات التنفيذ الخلفية (الرابعة لمسارات التنفيذ العادية) وهي عندما يموت مسار التنفيذ الرئيسي (الأول) للبرنامج.

مشاركة البيانات

مسارات التنفيذ التي أنشأناها من الفئة Thread هي مسارات تنفيذ حرة Free Threading، بحيث يمكن لكل مسار تنفيذ من مشاركة البيانات في نفس البرنامج العامل، وهذه الشيفرة خير دليل:

```
Public X As Integer ' متغير عام

Sub Main()
    Dim counter As Long
    Dim newThread As New Thread(AddressOf DoSomething)

    newThread.Start()

    For counter = 1 To 1000
        X += 1
    Next
    newThread.Join()

    Console.WriteLine("X = " & X) ' 2000
End Sub

Sub DoSomething()
    Dim counter As Integer

    For counter = 1 To 1000
        X += 1
    Next
End Sub
```

وضحت لك سابقا مشكلة من مشاكل مشاركة البيانات بين مسارات التنفيذ المتعددة، ودعني أذكرك هنا بان المشكلة اخطر من كونها مشاركة المتغيرات العامة، بل قد تشمل كل نشاطات الشيفرات المصدرية والكائنات الأخرى التي تستخدمها كمشاركة الملفات، الأجهزة والعتاد، مواقع الذاكرة، ....الخ. لذلك، ستضطر إلى استخدام **التزامن Synchronization** حتى تتقي شر التعارضات التي قد تحدثها لك مسارات التنفيذ المتعددة.

PDF

المتغيرات المحلية الديناميكية

جميع المتغيرات بكافة أنواعها (من منطلق قابليـــة رؤيتهـــا Visibility وعمرهـــا Lifetime) مشتركة بين مسارات التنفيذ، واعني بكلمة مشتركة –في هذا السياق- انها قابلة للوصول من جميع مسارات التنفيذ التي تعرفها في البرنامج. مع ذلـــك، المتغيـــرات المحليـــة الديناميكيـــة Local Dynamic Variables هي الوحيدة التي لا يتم تشاركها بين مسارات التنفيذ المتعددة، تحقق من هذه الشيفرة:

```
Sub Main()
    Dim newThread As New Thread(AddressOf DoSomething)
    Dim newThread2 As New Thread(AddressOf DoSomething)

    newThread.Start()
    newThread2.Start()

End Sub

Sub DoSomething()
    ' متغير محلي ديناميكي
    Dim X As Integer
    ' متغير محلي ديناميكي
    Dim counter As Integer

    Do
        counter += 1
        X += 1
    Loop Until counter >= 100

    Console.WriteLine(X)
End Sub
```

مخرجات الشيفرة السابقة ستكون دائما وأبدا:

```
100
100
```

لو كانت المتغيرات المحلية الديناميكية متشاركة بين مسارات التنفيذ المتعددة، لقام كلا المـــسارين السابقين بالتأثير على قيمة المتغير counter بحيث لا يتم تنفيذ الحلقة 100 مرة لكل مسار تنفيذ. فلو حاولت تعديل الإجراء DoSomething السابق، وجعلت المتغير counter ستاتيكي Static:

```
…
…
Sub DoSomething()
    ' متغير محلي ديناميكي
    Dim X As Integer
    ' متغير محلي ستاتيكي
    Static counter As Integer

    Do
        counter += 1
        X += 1
    Loop Until counter >= 100

    Console.WriteLine(X)
End Sub
```

فسيظهر لنا المتغير X (والذي اقصد به عدد مرات التكرار الحقيقية التي تمت للحلقة) القيمتين التاليتين:

```
100
1
```

إن سألتني عن سبب عدم تشارك المتغيرات المحلية الديناميكية بين مسارات التنفيذ المتعددة، فستكون إجابتي هي ان هذا النوع من المتغيرات يتم إنشائه لحظة التصريح عنها في داخل الإجراء عند استدعائه من قبل مسار التنفيذ، وسيتم حفظها بشكل مؤقت في ذاكرة Stack، مما يعني ان لكل مسار تنفيذ -يدخل الإجراء- نسخة خاصة من المتغيرات المحلية الديناميكية، لذلك لن يتم مشاركتها.

المواصفة ThreadStatic Attribute

إلى جانب المتغيرات المحلية الديناميكية، يمكنك استخدام المواصفة ThreadStatic Attributes على المتغيرات العامة أو على مستوى الوحدة لمنع مشاركتها بين مسارات التنفيذ المتعددة:

```
Module Module1
    Dim X As Integer
    <ThreadStatic()> Dim Y As Integer
    Sub Main()
        Dim newThread As New Thread(AddressOf DoSomething)

        X = 100
        Y = 100
        newThread.Start()
        newThread.Join()
```

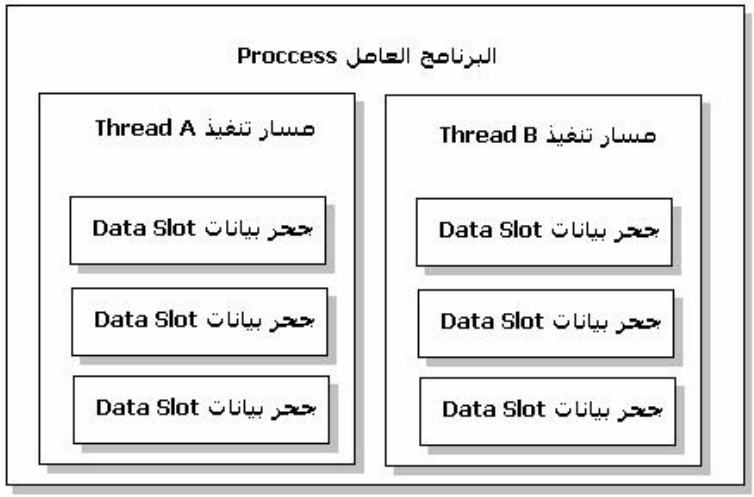
```
        Console.WriteLine(X) ' 200
        Console.WriteLine(Y) ' 100

    End Sub

    Sub DoSomething()
        X = 200
        Y = 200
    End Sub
End Module
```

قد يسرك كثيرا استخدام المواصفة ThreadStatic Attribute وتجنب عشرات المشاكل والأخطاء التي تحدث بسبب التعارض على المتغيرات من قبل مسارات التنفيذ المتعددة، ولكن يؤسفني إخبارك ان المواصفة ThreadStatic Attribute لا تعمل بشكل صحيح إلا مع المتغيرات المشتركة (اقصد هنا المتغيرات المعرفة في الوحدات البرمجية Modules أو في الفئات Classes باستخدام الكلمة المحجوزة Shared)، فعند تجربتي للشيفرة المصدرية التالية، أصبت بخيبة أمل كبيرة بعد رؤيتي لقيمة المتغير X:

```
Class TestClass
    ' لن يفيدك استخدام المواصفة هنا
    <ThreadStatic()> Public X As Integer
    ' Shared ستعمل بكفائة على هذا الحقل لانه مشترك
    <ThreadStatic()> Public Shared Y As Integer
End Class

Module Module1
    Dim TestObject As New TestClass()
    Sub Main()
        Dim newThread As New Thread(AddressOf DoSomething)

        TestObject.X = 100
        TestObject.Y = 100

        newThread.Start()
        newThread.Join()

        Console.WriteLine(TestObject.X) ' 200
        Console.WriteLine(TestObject.Y) ' 100

    End Sub

    Sub DoSomething()
        TestObject.X = 200
        TestObject.Y = 200
    End Sub
End Module
```

وحدة التخزين المحلية TLS

كل مسار تنفيذ تنشئه في شيفراتك المصدرية، يحمل معه **وحدة تخزين محلية Thread Local Storage (TLS)**، بحيث تمكن وحدة التخزين هذه مسار التنفيذ من حفظ بياناته الخاصة والغير قابلة للمشاركة مع مسارات تنفيذ أخرى. كل قيمة تحفظها في TLS تسمى **جحر بيانات Data Slot (شكل 4-10)**.



شكل 4-10: كل مسار تنفيذ يحتوي على مجموعة من جحور البيانات Data Slots.

يوفر لك إطار عمل .NET Framework أسلوبين للتعامل مع جحور البيانات Data Slots هما: **جحور البيانات المسماة Named data slots**، **وجحور البيانات غير المسماة Unnamed data slots**.

في جحور البيانات المسماة Named data slots، كل كائن أو قيمة تحفظها ترفق معها اسما حرفيا من النوع String يميزها ويكون كمعرف لها، يمكنك استخدام نفس هذا الاسم في كل مرة تود استرجاع القيمة. وحتى تتمكن من إنشاء جحر بيانات جديد، استدعي الطريقة AllocateNamedDataSlot():

```
' إنشاء جحر بيانات مسمى
Thread.AllocateNamedDataSlot("Name")
```

PDF Factory Pro trial version www.fastio.com

الطريقة السابقة تعود بكائن من النوع LocalDataStoreSlot، لذلك يفضل تعريف متغير من هذا النوع حتى يسهل عليك التعامل مع جحر البيانات لاحقا:

```
Dim namedDataSlot As LocalDataStoreSlot
namedDataSlot = Thread.AllocateNamedDataSlot("Name")
```

وحتى تتمكن من إسناد قيمة حقيقة لجحر البيانات المسمى Name السابق، استدعي الطريقة SetData():

```
Thread.SetData(namedDataSlot, "تركي العسيري")
```

وعلى العكس، تمكنك الطريقة GetData() من قراءة جحر بيانات مسمى:

```
Console.WriteLine(Thread.GetData(namedDataSlot)) ' تركي العسيري
```

وفي حال ما ان تم فقدان المؤشر nameDataSlot السابق، فلن تتمكن من تحديد جحر البيانات Name إلا باستخدام الطريقة GetNamedDataSlot():

```
Dim namedDataSlot2 As LocalDataStoreSlot
namedDataSlot2 = Thread.GetNamedDataSlot("Name")

Console.WriteLine(Thread.GetData(namedDataSlot2)) ' تركي العسيري
```

اخيرا، لا تنسى إلغاء جحر البيانات عند عدم الحاجة اليه باستدعاء الطريقة FreeNamedDataSlot():

```
Thread.FreeNamedDataSlot("Name")
```

ملاحظة

إن تم إرسال اسم جحر بيانات غير موجود في وحدة التخزين TLS إلى الطريقة GetNamedDataSlot()، فستقوم الطريقة بإنشاء جحر بيانات جديد.

PDF

اما النوع الثاني من جحور البيانات هو جحور البيانات غير المسماة Unnamed data slots (تسمى أيضا Allocated data store slots)، فهي مثل جحور البيانات المسماة السابقة تماما، إلا انها لا تستخدم معرفات حرفية من النوع String للكتابة والقراءة من وإلى جحر البيانات، بل ستكتفي باستخدام مؤشر من النوع LocalDataStoreSlot لتميز جحر البيانات:

```
Dim dataSlot As LocalDataStoreSlot
dataSlot = Thread.AllocateDataSlot()
Thread.SetData(dataSlot, "تركي العسيري")
Console.WriteLine(Thread.GetData(dataSlot)) ' تركي العسيري
```

تبادل البيانات بين مسارات التنفيذ

استخدام المتغيرات العامة Global Variables بين مسارات التنفيذ يعتبر أسلوب برمجي فاشل جدا بسبب مشكلة التعارضات التي تحدثنا عنها سابقا، كما ان استخدام وحدات التخزين المحلية TLS مباشرة أمر مستحيل لمشاركة البيانات (وذلك لان كل مسار تنفيذ له وحدة تخزين TLS خاصة به).

الحل الذي يمكننا من تبادل البيانات بين مسارات التنفيذ المختلفة برمجي بحت، فيمكنك مثلا البدء بتشغيل مسار تنفيذ بحيث ينتمي إلى إجراء تابع لفئة معينة وإرسال كافة البيانات المطلوبة إلى تلك الفئة، بحيث نحصر عملية تنفيذ شيفرات مسار التنفيذ في فئة معينة:

```
Class ThreadData
    Public ThreadName As String
    Public Data As Integer

    Sub DoSomething()
        Console.WriteLine(Me.ThreadName & " Data = " & Data)
    End Sub
End Class
```

الفئة ThreadData نريد منها ان تكون مستقلة بمسار التنفيذ الذي ينفذها، وعند إنشائك لكائن من هذه الفئة، يمكنك ارسال البيانات التي تودها إلى خصائص الفئة، والبدء بتشغيل مسار التنفيذ من الطريقة DoSomthing():

```
Module Module1
    Sub Main()
        Dim One As New ThreadData()
        Dim Two As New ThreadData()

        One.ThreadName = "THREAD B"
        One.Data = 10
        Two.ThreadName = "THREAD C"
        Two.Data = 20

        With New Thread((AddressOf One.DoSomething))
            .Start()
        End With
        With New Thread((AddressOf Two.DoSomething))
            .Start()
        End With
        Console.WriteLine("Waiting...")
    End Sub
End Module
```

عند تنفيذي للشيفرة السابقة، ظهرت المخرجات بهذا الشكل:

```
Waiting…
THREAD B Data = 10
THREAD C Data = 20
```

مع ذلك، فترتيب المخرجات لا يشترط ان يكون متوافقا عند تنفيذها على جهاز آخـــر، حيـــث ان التسلسل قد يختلف على حسب مزاج المعالج، وقد تكون مخرجاتك بهذا الشكل:

```
THREAD C Data = 20
Waiting…
THREAD B Data = 10
```

كان الغرض من الفئة ThreadData السابقة هو جعل مسار التنفيذ يعمل في كائن مستقل، بحيث لا تؤثر شيفراته المصدرية على بيانات مسارات التنفيذ الأخرى في البرنامج. ولكن توجـــد نقطة هامة عليك أخذها بعين الاعتبار تتعلق بالأحداث Events التابعة للفئات، فلو أضفت الحدث التالي في الفئة ThreadData:

```
Class ThreadData
    …
    …
    Event ThreadFinish()

    Sub DoSomething()
        …
        …
        RaiseEvent ThreadFinish()
    End Sub
End Class
```

وحاولت قنصه من مسار التنفيذ الرئيسي للبرنامج:

```
Module Module1
    Sub Main()
        …
        …
        AddHandler One.ThreadFinish, AddressOf ThreadHasBeenFinished
        …
        …
    End Sub

    Sub ThreadHasBeenFinished()
        …
        …
    End Sub
End Module
```

فتأكد وثق ثقة تامة ان الإجراء ()ThreadHasBeenFinished السابق سيتم تنفيذه من قبل مسار التنفيذ الذي أدى إلى إطلاق الحدث وليس من قبل مسار التنفيذ الرئيسي (بالرغم من ان عملية القنص باستخدام AddHandler تمت من قبل مسار التنفيذ الرئيسي). لذلك، احرص على اخذ هذه المسالة بعين الاعتبار عند إطلاق الأحداث.

التزامن Thread Synchronization

في هذا القسم أحاول عرض لك بضعة أساليب تمكنك من تزامن مسارات التنفيذ على الشيفرات المصدرية حتى تتقي شر التعارضات.

التركيب SyncLock ... End SyncLock

ان كنت من المبتدئين في برمجة مسارات التنفيذ المتعددة، فقد يأتيك شعور بان كل سطر من شيفرات البرنامج المصدرية سيتم تنفيذه بالكامل من قبل مسار تنفيذ معين وهو -مع الأسف الشديد- شعور خاطئ تماما، فلو كان لدينا الشرط التالي:

```
If X = 10 Then
    X = 0
Else
    X = 10
End If
```

وتم تنفيذه من قبل مسارين تنفيذين، فلا تعتقد ان مسار التنفيذ الأول سيتحقق من الشرط ومن ثم يقوم بتنفيذ جواب الشرط بينما مسار التنفيذ الثاني يتفرج عليه كالأطرش في الزفة، بل ستحدث التعارضات بين مسارات التنفيذ أيضا في اصغر جزء من أجزاء شيفراتك البرمجية.

من هنا يأتي دور التركيب SyncLock ... End SyncLock والذي يمكنك من اعتبار جزء معين من شيفراتك المصدرية كوحدة واحدة بحيث يتم تنفيذها من قبل مسار تنفيذ واحد فقط، لتبقى مسارات التنفيذ الأخرى في قائمة الانتظار إن أرادت تنفيذ الشيفرة الموجودة في داخل هذا التركيب.

كل ما تحتاجه لاستخدام التركيب SyncLock ... End SyncLock في شيفراتك المصدرية، متغير مرجعي Reference Type يحمل أي قيمة لا تساوي Nothing ويكون مشترك بين مسارات التنفيذ الأخرى:

```
SyncLock y
    If X = 10 Then
        X = 0
    Else
        X = 10
    End If
End SyncLock
```

والان اضمن لك ان الشيفرة الموجودة داخل التركيب السابق سيتم تنفيذها من قبل مسار تنفيذ واحد فقط، وفي حال ما وجد مسار تنفيذ آخر يستخدم نفس الشيفرة، فسينتظر في طابور ريثما ينهي مسار التنفيذ الحالي إكمال الأجزاء المتبقية من الشيفرة.

حتى اريك تطبيق عملي ذو تأثير ظاهري على هذا التركيب، راقب معي هذه الشيفرة:

```
Module Module1
    Sub main()
        Dim counter As Integer

        For counter = 1 To 9
            With New Thread(AddressOf DoSomething)
                .Name = CStr(counter)
                .Start()
            End With
        Next
        Console.Read()
    End Sub

    Sub DoSomething()
        Dim counter As Integer


        Console.WriteLine("From Thread " & _
            Thread.CurrentThread.Name)

        For counter = 1 To 1000
            Console.Write(Thread.CurrentThread.Name)
        Next
        Console.WriteLine()
    End Sub
End Module
```

مخرجات الشيفرة السابقة ستكون غير واضحة بشكل كبير، حيث سيتم تنفيذ الحلقة من قبل أكثر من مسار تنفيذ في نفس الوقت وتكون المخرجات شيئا مثل:

```
From Thread 1
111111111111111111111111111111111111111 From Thread 2
2222222222222222222222222222211111111111111111111 From Thread 7
777111112222222222221111111
…
…
```

والان لنعدل الإجراء DoSomething() ونجعل شيفراته داخل تركيب SyncLock … End SyncLock:

PDF

```
' متغير مرجعي عام ومشترك بين كافة مسارات التنفيذ
Public ThreadLocker As String = "Lock1"
…
…

Sub DoSomething()
    Dim counter As Integer

    SyncLock ThreadLocker
        Console.WriteLine("From Thread " & _
            Thread.CurrentThread.Name)
        For counter = 1 To 1000
            Console.Write(Thread.CurrentThread.Name)
        Next
        Console.WriteLine()
    End SyncLock
End Sub
```

بعد إضافة التركيب السابق، سيضمن لك Visual Basic .NET بان عملية تنفيذ الشيفرات الموجودة داخل هذا التركيب تتم من قبل مسار تنفيذ واحد فقط، وبقية مسارات التنفيذ ستظل في طابور انتظار، لذلك ستكون المخرجات أكثر تنظيما:

```
From Thread 1
1111111111111111111111111111111111111111111111111111 …
From Thread 5
5555555555555555555555555555555555555555555555555555 …
From Thread 3
3333333333333333333333333333333333333333333333333333 …
From Thread 2
2222222222222222222222222222222222222222222222222222 …
…
…
```

المواصفة Synchronization

بدلا من استخدامك للتركيب SyncLock … End SyncLock عشرات المرات، يمكنك استخدام المواصفة Synchronization في أعلى الفئة حتى تجعل جميع الشيفرات التي بداخلها يتم تنفيذها في قبل مسار تنفيذ واحد فقط:

```
<System.Runtime.Remoting.Contexts.Synchronization()> _
Class Test
    ' ContextBoundObject لابد من اشتقاق الفئة
    Inherits ContextBoundObject
    …
    …
End Class
```

PDF

يعيب المواصفة Synchronization هو ان الفئة التي تستخدمها لابد من تكـــون مـــشتقة وراثيا من الفئة ContextBoundObject مما يحرمك من إمكانية اشتقاق فئة أخرى. عيب آخر في المواصفة Synchronization يتعلق بالأعضاء المشتركة Shared Members، حيث يمكن ان تحدث التعارضات على الشيفرات المصدرية في أعضاء الفئة المشتركة من قبل مسارات التنفيذ المتعددة. مع ذلك، تستطيع الالتفاف حول هذه العيوب باستخدام المواصفة MethodImpl.

المواصفة MethodImpl

استخدامك للمواصفة Synchronization يمنع جميع مسارات التنفيذ من تنفيذ شيفرة الكـــائن ان كان مشغول بأحد مسارات التنفيذ، وهذا بحد ذاته أسلوب غير مرن، حيث ما لفائـــدة مـــن تعـــدد مسارات التنفيذ إن كانت العمليات لا تتم إلا من قبل مسار تنفيذ واحد فقط في وقت واحد.

استخدامك للمواصفة MethodImpl يمكنك من تحديد الأعضاء التي تود حكرها على مسار تنفيذ واحد، مع العلم انها تعمل بنجاح أيضا على الأعضاء المشتركة Shared Members. هـــذه طريقة استخدامها:

```
' استرد مجال الأسماء التالي لاختصار الشيفرة
Imports System.Runtime.CompilerServices
…
…
Class Test1
    <MethodImpl(MethodImplOptions.Synchronized)> _
    Sub AA(ByVal x As Char)
        …
        …
    End Sub

    <MethodImpl(MethodImplOptions.Synchronized)> _
    Shared Sub BB(ByVal x As Char)
        …
        …
    End Sub

    …
    …
End Class
```

بودي توضيح بعض المسائل المترتبة على هذه المواصفة، حيث ان عملية الإقفـــال لا تـــتم على الطريقة التي عرفت فيها المواصفة فقط، بل تشمل وتمتد إلى الطرق الأخـــرى، فمـــثلا لـــو استخدمت المواصفة في هاتين الطريقتين:

```
Class Test2
    <MethodImpl(MethodImplOptions.Synchronized)> _
    Sub AA(ByVal x As Char)
        …
        …
    End Sub
    <MethodImpl(MethodImplOptions.Synchronized)> _
    Sub BB(ByVal x As Char)
        …
        …
    End Sub

    …
    …
End Class
```

فاعلم ان الشيفرات المصدرية لكلا الطريقتين سيتم تنفيذها من قبل مسار تنفيذ واحد فقط وليس من قبل مسار تنفيذ واحد فقط للطريقة الأولى وآخر للطريقة الثانية، أي –كمزيد من التوضيح- ان كان مسار التنفيذ الأول يجري عمليات الطريقة ()AA، فان مسار التنفيذ الثاني سيضطر إلى الانتظار رغم انه يود تنفيذ شيفرات الطريقة الآخر ()BB.

اما مع الأعضاء المشتركة (كالفئة Test1 السابقة) فيمكن ان يتم تنفيذها من قبل مسار تنفيذ آخر ان كان مسار التنفيذ يجري عمليات احد الأعضاء العادية Instance Members (الغير مشتركة).

فئات أخرى

إلى جانب التركيب SyncLock … End SyncLock والمواصفتين Synchronization و MethodImpl، توجد مجموعة إضافية من الفئات والخاصة بتزامن عمل مسارات التنفيذ المتعددة تستخدم في اغلب البرامج الجديدة هي: Monitor، Interlocked، Mutex، ReaderWriterLock، ManualResetEvent، و AutoResetEvent. يمكنك البحث عن تفاصيل استخدامها في مراجع NET Documentation. ان كنت مهتما في هذا الموضوع، وذلك لأني لن اعرض لك هنا إلا الفئة الأولى Monitor بشكل مبسط حيث أنها تمثل البنية التحتية للتركيب SyncLock … End SyncLock السابق ذكره.

يعيب التركيب SyncLock … End SyncLock انك تحدد الشيفرة المصدرية وقت التصميم وليس التنفيذ، فلن تستطيع –مثلا- حصر مجموعة من الشيفرة المصدرية في كتلة واحدة ان تم تحقق شرط معين لحظة التنفيذ. ومن هنا يأتي دور الفئة Monitor والتي تقوم بنفس عمل التركيب SyncLock … End SyncLock ولكن في وقت التنفيذ وليس التصميم.

PDF Factory Pro trial version www.fastio.com

لست بحاجة لإنشاء كائن جديد من الفئة Monitor لاستخدامها، حيث ان جميع طرقها مشتركة Shared Methods، استدعي الطريقة Enter() لتحديد نقطة البداية للشيفرة المراد حصرها، والطريقة Exit() لتحديد نقطة النهاية، مع العلم ان كلا الطريقتين تتطلبان متغير عام Global Variables مرجعي Reference قيمته لا تساوي Nothing –تماما كمتطلبات التركيب SyncLock ... End SyncLock - فلو كان لدينا هذا التركيب:

```
Public X As String = "Lock1"
…
SyncLock X
    …
    ' الشيفرة المراد حصرها على مسار تنفيذ واحد فقط
    …
End SyncLock
```

يمكنك تحويله إلى الفئة Monitor بهذا الشكل:

```
Public X As String = "Lock1"
…
Monitor.Enter(X)
    …
    ' الشيفرة المراد حصرها على مسار تنفيذ واحد فقط
    …
Monitor.Exit(X)
```

وكنوع من المرونة التي توفرها لك الفئة Monitor، يمكنك تنفيذه داخل جملة شرطية:

```
Public X As String = "Lock1"
…
If Y = 0 Then Monitor.Enter(X)
…
' الشيفرة المراد حصرها على مسار تنفيذ واحد فقط
…
Monitor.Exit(X)
```

ملاحظة

استدعائك للطريقة Exit() دون الطريقة Enter() سيظهر رسالة خطأ. لذلك، عليك تصحيح الشائب في الشيفرة السابقة في حال ان لم تكن قيمة الشرط صحيحة:

If Y = 0 Then Monitor.Exit (X)

PDF

المزيد أيضا، الشيفرة المحصورة بين فكي الاستدعائين Enter() و Exit() ستجعل كل مسارات التنفيذ الأخرى في قائمة الانتظار حتى ينهي مسار التنفيذ الحالي تنفيذ الشيفرة، مع ذلك قد تود من مسارات التنفيذ الأخرى الانتظار لفترة من الوقت تحددها كوسيطة ثانية إلى الطريقة TryEnter():

```
Public X As String = "Lock1"
…
' أنتظر فترة نصف ثانية ثم نفذ الشيفرات حتى
' لو كان مسار تنفيذ آخر يجري التنفيذ
Monitor.TryEnter(X, 500)
    …
    ' الشيفرة المراد حصرها على مسار تنفيذ واحد فقط
    ' لفترة اقل من نصف ثانية
    …
Monitor.Exit(X)
```

الفئة ThreadPool

إنشاء عشرات مسارات التنفيذ في برنامج عامل واحد يؤدي –بلا شك- إلى إضعاف كفاءة التنفيذ، كما انك في كل مرة تنشئ فيها مسار تنفيذ جديد (باستخدام الفئة Thread) ستجرى آلاف التعليمات والخاصة بنظام التشغيل لخلق المسار، ولحظة موت مسار التنفيذ تؤدي إلى آلاف العمليات –والخاصة بنظام التشغيل أيضا- لقتل المسار.

يمكن زيادة كفاءة التنفيذ بالاعتماد على **بركة مسارات التنفيذ Thread Pool**، حيث تمكنك من استعارة مسار التنفيذ –بشكل سريع- واستخدامه في برنامجك. يمكنك استعارة مسار تنفيذ باستدعاء الطريقة QueueUserWorkItem()، والتي تتطلب من الإجراء المفوض لها ان يحتوي على وسيطة واحدة من النوع Object:

```
ThreadPool.QueueUserWorkItem(AddressOf DoSomething)
…
…
' لابد ان يحتوي الاجراء المفوض على وسيطة من النوع Object
Sub DoSomething(ByVal x As Object)
    …
    …
End Sub
```

الميزة في الإجراء المفوض بمشيد الفئة ThreadPool، انك تستطيع إرسال أي قيمة له تستفيد منها، وهي ميزة حرمتنا منها سابقا الفئة Thread:

PDF Factory Pro trial version www.fastio.com

```
ThreadPool.QueueUserWorkItem(AddressOf DoSomething, 100)
…
…
Sub DoSomething(ByVal x As Object)
    …
    Console.WriteLine (X) ' 100
End Sub
```

أول استدعاء للطريقة QueueUserWorkItem() سيؤدي إلى إنشاء بركة مسارات تنفيذ تستوعب –بشكل افتراضي- على 25 مسار تنفيذ متساوية الأسبقية Priority، يمكنك معرفة العدد الأقصى من مسارات التنفيذ الممكن استعارتها في بركة مسارات التنفيذ بالاستعلام عنها في الطريقة GetMaxThreads()، كما تستطيع معرفة عدد مسارات التنفيذ المتوفرة في البركة لاستعارتها باستدعاء الطريقة GetAvailableThreads()، كلا الطريقتين تستقبل وسيطتين بالمرجع ByRef:

```
Dim x, y, z As Integer

ThreadPool.GetMaxThreads(x, z)
ThreadPool.GetAvailableThreads(y, z)

Console.WriteLine(x) ' 25
Console.WriteLine(y) ' 23
```

ملاحظة

الوسيطة الثانية لكلا الطريقتين GetMaxThreads() و GetAvailableThreads() يسند لها قيمة تمثل عدد مسارات التنفيذ الخاصة بدخل وخرج الملفات IO. راجع مكتبة MSDN لتفاصيل أكثر حول هذا الموضوع.

نقطة هامة جدا جدا، مسارات التنفيذ المستعارة من بركة مسارات التنفيذ هي مسارات تنفيذ متباعدة Apartment Threading وليس حرة Free Threading، لذلك كل مسار تنفيذ سيواجه نسخة خاصة به من المتغيرات العامة، الشيفرة المسطورة في أعلى الصفحة المقابلة خير دليل:

PDF

```
Module Module1
    Dim x As Integer
    Sub main()
        x = 100
        ThreadPool.QueueUserWorkItem(AddressOf DoSomething, Nothing)

        Thread.Sleep(1000)
        Console.WriteLine(x) ' 100
        Console.Read()
    End Sub

    Sub DoSomething(ByVal x As Object)
        x = 1000
        Console.WriteLine(x) ' 1000
    End Sub
End Module
```

مع ذلك، المتغيرات العامة المرجعية Global Reference Variables لن يتم إنشاء نسخة لهـــا لكل مسار تنفيذ، وذلك لأنها تحفظ في نفس ذاكرة Managed Heap:

```
Class TestClass
    Public x As Integer
End Class

Module Module1
    Public TestObject As New TestClass()
    Sub main()
        TestObject.x = 100
        ThreadPool.QueueUserWorkItem(AddressOf DoSomething, Nothing)

        Thread.Sleep(1000)
        Console.WriteLine(TestObject.x) ' 1000
        Console.Read()
    End Sub

    Sub DoSomething(ByVal x As Object)
        TestObject.x = 1000
    End Sub
End Module
```

أخيرا وليس آخرا، يمكنك معرفة ما اذا كان مسار التنفيذ الحالي يعمل في بركة مـــسارات التنفيذ بالاستعلام عن الخاصية IsThreadPoolThread:

```
If Thread.CurrentThread.IsThreadPoolThread Then
    …
    …
End If
```

هذا مثال كامل لاستخدام بركة مسارات التنفيذ:

```
Module Module1
    Sub main()
        Dim counter As Integer

        ThreadPool.QueueUserWorkItem(AddressOf DoSomething, 1)
        ThreadPool.QueueUserWorkItem(AddressOf DoSomething, 2)
        ThreadPool.QueueUserWorkItem(AddressOf DoSomething, 4)
        Console.Read()
    End Sub

    Sub DoSomething(ByVal x As Object)
        Dim counter As Integer
        For counter = 1 To 100000
            Console.Write(x)
        Next
    End Sub
End Module
```

ايهما أفضل، إنشاء مسارات التنفيذ مباشرة باستخدام الفئة Thread أو استعارة مسار تنفيذ من برك مسارات التنفيذ باستخدام الفئة ThreadPool؟ اجابتي -كل العادة- ستكون معتمدة على متطلباتك، ولكن دعني اذكرك هنا بان استعارة مسارات التنفيذ من البركة لا تقوم بتنفيذ الاجراء المفوض لها فورا كما تفعل مع الفئة Thread، اضف إلى ذلك انها مسارات تنفيذ خلفية Background Thread، وان جعلتها عادية (باسناد القيمة False إلى الخاصية IsBackground) فلن تتمكن من قتلها باستخدام الطريقة Abort(). باختصار، أنشي مسارات التنفيذ من الفئة Thread ان كانت المسارات تعمل في فترة طويلة من عمر البرنامج دون توقف، وعود نفسك على الاستعارة من بركة مسارات التنفيذ ان كنت تنوي إنشاء وقتل الكثير من مسارات التنفيذ مرات عديدة لإنجاز مهام معينة في فترات معينة من حياة البرنامج.

المؤقتات Timers

يوفر لك إطار عمل .NET Framework. ثلاث انواع من **المؤقتات Timers** هي: Timers.Timer، Threading.Timer، و Windows.Forms.Time تجدها في مجال الأسماء الجذري System. في هذا القسم اعرض لك المؤقتين الأول والثاني، اما الاخير فأرى انه من المناسب تأجيله حتى الفصل الرابع عشر **الأدوات Controls**.

PDF Factory Pro trial version www.fastio.com

المؤقت System.Timers.Timer

عند إنشائك لكائن من النوع Timers.Timer، عليك قنص حدثه Elapsed (باستخدام WithEvents أو AddHandler) والذي يتم تفجيره كل فترة من الوقت تحددها في الخاصية Interval أو إرسالها كمشيد:

```
' ثانية
Dim Timer1 As New Timers.Timer (1000)

' نصف ثانية
Dim Timer2 As New Timers.Timer ()
Timer2.Interval = 500
```

يمكنك اسناد القيمة False إلى الخاصية AutoReset ان اردت تفجير الحدث Elapsed مرة واحدة فقط، كما تستطيع تشغيل المؤقت باستدعاء الطريقة Start() وإيقافه في أي وقت بالطريقة Stop().

هذا مثال لاستخدام المؤقتات من النوع System.Timers.Timer:

```
Module Module1
    Sub main()
        Dim MyTimer As New System.Timers.Timer()
        AddHandler MyTimer.Elapsed, AddressOf DoSomething

        MyTimer.AutoReset = True
        MyTimer.Interval = 1000
        MyTimer.Start()

        Console.Read()
    End Sub

    Sub DoSomething(ByVal sender As Object, ByVal e As
Timers.ElapsedEventArgs)
        Console.WriteLine(Now.ToString("hh:mm:ss"))
    End Sub
End Module
```

ملاحظة

الإجراءات التي تم قنصها للحدث Elapsed والتابعة للمؤقتات من النوع System.Timers.Timer تعمل في بركة مسارات التنفيذ، يمكنك التحقق من ذلك:

```
Console.WriteLine(Thread.CurrentThread.IsThreadPoolThread) ' True
```

المؤقت System.Threading.Timer

يعمل المؤقت Threading.Timer كعمل المؤقت السابق Timers.Timer ولكنه يختلف في طريقة استخدامه، حيث يتطلب المؤقت Threading.Timer إجراء مفوض Delegate (وليس حدث Event) يتم تنفيذه كل فترة معينة.

إن أردت إنشاء كائن من الفئة Threading.Timer، عليك إرفاق جميع البيانات المطلوبة كوسيطات لمشيده. الوسيطة الأولى تمثل الإجراء المفوض –والذي يشترط ان يحتوي على وسيطة من النوع Object:

```
Dim MyTimer As New Threading.Timer(AddressOf DoSomething, …)

Sub DoSomething(ByVal x As Object)
    …
    …
End Sub
```

يمكنك ارسال أي قيمة إلى هذا الإجراء المفوض، بذكرها كوسيطة ثانية لمشيد فئة المؤقت:

```
' 100 ارسل القيمة
Dim MyTimer As New Threading.Timer( …, 100, …)

Sub DoSomething(ByVal x As Object)
    Console.WriteLine(x) ' 100
End Sub
```

اما الوسيطة الثالثة فتحدد فيها المدة التي تود بدء عملية تنفيذ الإجراء، والوسيطة الرابعة تحدد فيها الفترة الزمنية لتكرار تنفيذ الإجراء:

```
' نفذ الاجراء بعد 5 ثواني لمرة واحدة فقط
Dim MyTimer As New Threading.Timer( …, …, 5000, 0)

' نفذ الاجراء بعد 10 ثواني، وثم كرر التنفيذ كل نصف ثانية
Dim MyTimer As New Threading.Timer( …, …, 10000, 500)
```

لا توجد خصائص تمكنك من تعديل القيم السابقة، ولكن توجد الطريقة Change() التي تمكنك من تعديل قيمة الوسيطة الثالثة والرابعة للمشيد. اخيرا، تستطيع إيقاف عمل المؤقت باستدعاء الطريقة Dispose().

هذا مثال لاستخدام المؤقت Threading.Timer:

```
Module Module1
    Sub main()
        Dim MyTimer As New Threading.Timer(AddressOf DoSomething, _
        Nothing, 1, 2000)

        Console.Read()
    End Sub

    Sub DoSomething(ByVal x As Object)
        Console.WriteLine(Now.ToString("hh:mm:ss"))
    End Sub
End Module
```

ملاحظة

الإجراءات المفوضة للمؤقتات من النوع System.Threading.Timer تعمل أيضا في بركة مسارات التنفيذ.

PDF

أكاد لا اصدق أني انتهيت فعلا من كتابة هذا الفصل والخاص بمسارات التنفيذ Threading، إن كان الشرح غير واضح ولم تفهم منه شيئا، فتذكر ان برمجة مسارات التنفيذ المتعددة Multi-Threading من اعقد وأصعب المواضيع البرمجية والتي شهدتها أنامل المبرمجين على مر التاريخ. أما الصفحات المقبلة ستأخذك إلى موضوع جديد كليا وحديث العهد، اذ انه ظهر مع إطار عمل .NET Framework فقط وهو **المجمعات Assemblies**.

PDF Factory Pro trial version www.fastio.com

المجمعات Assemblies

سواء كنت تنوي تطوير تطبيقات Windows قياسية، أو مكتبات فئات Class Libraries، أو أي نوعية أخرى من البرامج، عليك معرفة ان التطبيق الذي تطوره وتنوي توزيعه يسمى **مجمع Assembly**.

في هذا الفصل النظري أتحدث عن مجموعة كبيرة من المواضيع المتفرقة والتي تظهر لك البنية التحتية لتركيبة المجمعات، كما سنستخدم مجموعة من أدوات الربط والترجمة لتمكنك من دمج شيفرات مكتوبة بلغات NET. مختلفة في برنامج واحد. ولكن قبل ذلك، دعنا نبدأ من الأساس وتوضيح الوحدات المدارة Managed Modules.

الوحدات المدارة Managed Modules

عندما تقوم بترجمة برنامجك المكتوب بـ NET. Visual Basic (أو أي لغة NET. اخرى)، فالنتيجة النهائية تسمى **وحدة مدارة Managed Module**، والتي قد تكون **في** ملف EXE أو DLL (لاحظ حرف الجر "**في**"، فالملف التنفيذي EXE قد يشمل أكثر موحدة مدارة)، هذه الوحدة المدارة يمكن ان تعمل بشكل مستقل أو تعتمد على وحدات مدارة اخرى ليتم تنفيذها. الوحدة المدارة أو مجموعة الوحدات المدارة المترابطة تكون ما يسمى **المجمع Assembly**. إذاً، فالمجمع قد يحتوي على وحدة مدارة واحدة أو مجموعة من الوحدات المدارة.

تتكون الوحدة المدارة من اربعة اقسام هي:

1) **Windows PE File Header**: وهو رأس للملف خاص بنظام التشغيل Windows والذي تشمله جميع التطبيقات التي تعمل تحت بيئة Microsoft Windows.

2) **NET Framework File Header.**: وهو أيضا رأس للملف خاص بالتطبيقات الموجهة إلى إطار عمل NET. Framework. يحتوي على معلومات حول

الوحدة المدارة الحالية كنقطة بداية التنفيذ، مؤشرات إلى شيفرات التنفيذ، أو إلى بيانات Metadata وغيرها...

3) **Metadata**: يحتوي هذا القسم على وصف لأنواع البيانات الموجودة في الوحدة المدارة (كالفئات Classes، الوحدات البرمجية Modules، التركيبات من النوع Structure أو Enum، الواجهات Interfaces ...الخ). يستخدم إطار عمل NET. هذا القسم للتحقق من أنواع البيانات المرسلة بين المجمعات أو الوحدات المدارة المختلفة، كما يمكنك الاستعلام عن هذه البيانات باستخدام مجموعة من فئات الانعكاس Reflection Classes كما سترى لاحقا في الفصل القادم.

4) **شيفرة MSIL**: كما ذكرت في الفصل الأول **تعرف على Visual Basic NET.** وبالتحديد عن فقرة **الترجمة على الفور JIT**، حيث قلت ان شيفراتك المصدرية لحظة الترجمة سيتم تحويلها إلى لغة شبيهه بلغة التجميع Assembly تسمى Microsoft Intermediate Language (MSIL أو IL). شيفرة MSIL تمثل شيفرة كل أجزاء الوحدة المدارة بعد الترجمة.

تطبيقيا، لن تستفيد الكثير من القسم الاول، الثاني، والرابع الا ان كنت تنوي التخصص في البنية التحتية لتطبيقات Windows، ولغة التنفيذ المشتركة CRL الخاصة بإطار عمل NET. وهي مواضيع متقدمة جدا وخارج نطاق الكتاب، اما القسم الثالث Metadata فسترى في الفصل القادم **فئات الانعكاس Reflection Classes** كيف يمكنك الاستفادة منها والاستعلام عن البيانات الموجودة في الوحدة المدارة بـ NET. Visual Basic، لتتمكن –مثلا- من معرفة جميع الفئات والاستعلام عن جميع أعضائها من برنامج خارجي اخر.

المجمعات Assemblies

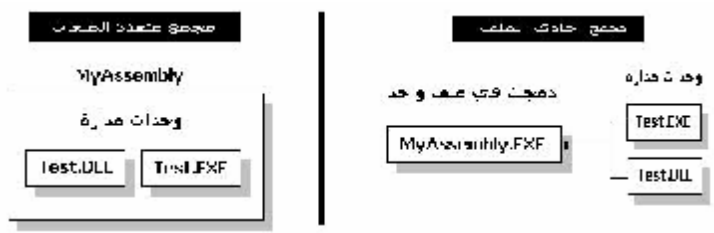
المجمع Assembly هو اصغر وحدة تمثل تطبيق منجز بأحد لغات NET.، أي بعبارة اخرى المجمع هو التطبيق الذي أنشأته، بحيث يكون له اسم خاص، رقم إصدار، اسم المصمم، الشركة ...الخ. قد يكون المجمع في ملف قابل للتنفيذ مباشرة EXE، أو مكتبة DLL تستخدمها في برامج اخرى، أو مجموعة مترابطة من الملفات (EXE، DLL، DOC، HTML، BMP ...الخ). يحتوي المجمع على مجموعة من العناصر –كما ذكرت في الفقرة السابقة- تسمى الوحدات المدارة. ليس هذا فقط، بل قد يحتوي المجمع على عناصر اخرى ليست تنفيذية Nonexecutable

كملفات المصادر Resource Files، صفحات HTML، ملفات نصوص Texts، صور Pictures ... الخ.

لذلك، عندما يتباهى مبرمجو NET. فيما بينهم، فإنهم يستخدموا المصطلح أنجزنا مجمعات عوضا عن المصطلحات برامج أو تطبيقات.

المجمعات الأحادية والمتعددة الملفات

لا يشترط ان تكون جميع الوحدات المدارة والتابعة لمجمع معين مدمجة في ملف واحد، بل يمكن ان يكون المجمع مجموعة من الملفات المستقلة، مع ذلك جميع هذه الملفات مترابطة –نظريا- وكأنها ملف واحد لتكون المجمع (**شكل 11-1**). فلو كان للمجمع صلاحيات معينة، فان هذه الصلاحيات تشمل جميع الوحدة المدارة التابعة له، وان كان للمجمع رقم إصدار خاص فستحمل كافة الوحدات المدارة نفس هذا الرقم.



شكل 11-1: المجمعات أحادية ومتعددة الملفات

معظم التطبيقات التي أنجزناها في هذا الكتاب تحتوي على وحدة مدارة واحدة، وبعد عملية الترجمة يمثل الملف التنفيذي EXE مجمع أحادي الملف، تمكنا في الفصول السابقة من إنجاز هذه المجمعات باستخدام بيئة التطوير Visual Studio .NET وبالتحديد بعد الضغط على المفتاح [F5]. اما ان كان لديك أكثر من وحدة مدارة وأردت دمجها في ملف واحد، فعليك استخدام الأداة AL.EXE والتي سيأتيك تفصيلها لاحقا.

اعيد واكرر نقطة هامة عليك أخذها دائما في الاعتبار، المجمعات سواء كانت أحادية الملفات أو متعددة الملفات هي برنامج واحد، ويمكنك تخيله –منطقيا- كملف EXE أو DLL واحد رغم انه –فيزيائيا- متعدد الملفات.

أساليب تنفيذ المجمعات

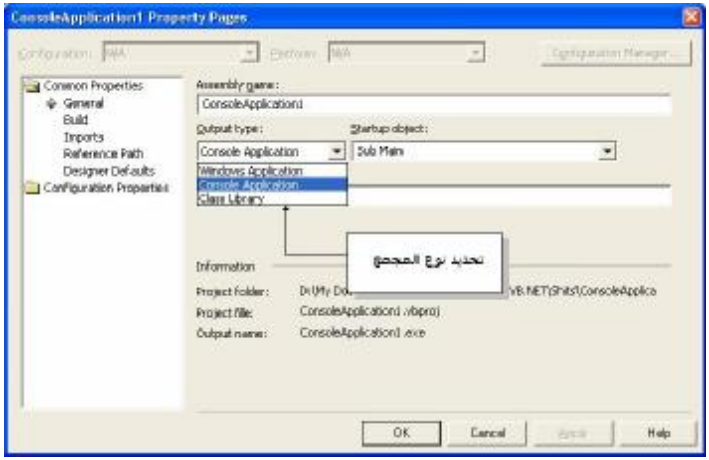
المجمعات (البرامج والتطبيقات) مهما كان غرضها ومهما كانت طرق إنجازها، يتم تنفيذها -في عالم .NET- بطريقة واحدة من ثلاثة طرق هي: مكتبة Library، تطبيق Windows Application، أو تطبيق Console Application.

بالنسبة للمكتبات Libraries فيكون امتداد الملف النهائي للمجمع في اغلب الأحوال DLL، لن تستطيع تشغيلها مباشرة لاستخدامها، وانما عليك إضافتها في خانة المراجع Reference حتى تتمكن من الاستفادة من فئاتها في برامجك الأخرى والتي تنجزها بلغات .NET الأخرى.

اما التطبيقات Windows Application و Console Application يمكن تشغيلها مباشرة، لذلك هي تتطلب منك **نقطة إدخال Entry Point**. وهي تمثل الإجراء الابتدائي (يكون Sub Main() في اغلب تطبيقات .NET Visual Basic) الذي يتم استدعائه لحظة تنفيذ البرنامج.

التطبيقات Windows Application و Console Application متشابهة إلى حد كبير، ويكمن الفرق بينهما ان النوع الثاني يمكنك من استخدام الكائن Console والخاص بعرض المخرجات، اما الاول فلا يعتمد عليه.

يمكنك تحديد نوع المجمع من خانة Output Type في صندوق الحوار Project Property Pages (**شكل 11-2**).



شكل 11-2: تحديد نوع المجمع.

المجمعات الخاصة والمشتركة

يوجد نوعان من المجمعات يمكنك إنجازها هما **المجمعات الخاصة Private Assemblies والمجمعات المشتركة Shared Assemblies**. المجمعات الخاصة هي مجمعات تضع ملفاتها في نفس مجلد البرنامج أو المجلدات الفرعية التابعة له. ليس هذا فقط، بل إن استخدامها محصور للبرنامج الذي في نفس المجلد أو المجلد الأبوي لها –فهي خاصة به. فلو إفترضنا هذه المجمعات:

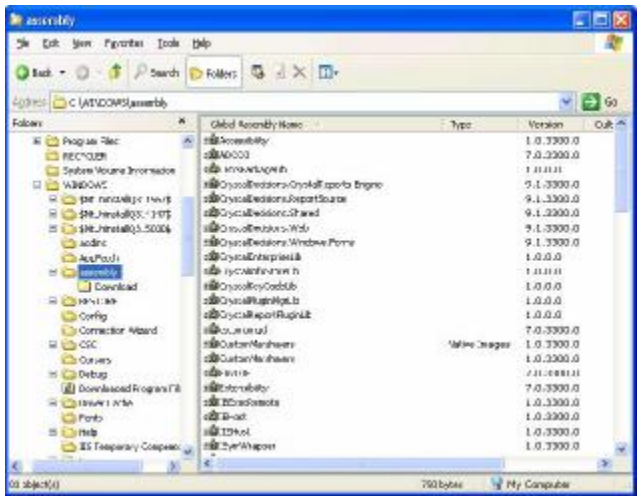
```
C:\Folder\AAA.DLL
C:\BBB.EXE
C:\Folder\CCC.EXE
C:\Folder\SubFolder\DDD.EXE
```

فيمكنك استخدام المجمع AAA.DLL من خلال التطبيقين BBB.EXE و CCC.EXE فقط، وذلك لانهما في نفس مجلد المجمع الخاص أو في المجلد الأبوي له. مع ذلك، عليك استخدام الوسم <probing> في ملف التهيئة للتطبيق BBB.EXE لتحدد فيهما اسم المجلد الفرعي والذي يحتوي على المجمع لانجاز ما يعرف **بالإيجاد Probing** (راجع مكتبة MSDN لطريقة استخدام هذا الوسم.

ملاحظة

عندما تقوم بإضافة مجمع مشترك في برنامجك عن طريق صندوق الحوار Add References، ستقوم بيئة التطوير Visual Studio .NET – مشكورة- بنسخ ملف هذا المجمع إلى مجلد برنامجك الرئيسي. مع ذلك، عندما تحذف المجمع من صندوق الحوار السابق، ستقوم بيئة التطوير –غير مشكورة هذه المرة- بحذف ملف المجمع من مجلد البرنامج مباشرة **دون سابق إنذار** !

اما المجمعات المشتركة فهي مجمعات قابلة للاستخدام من قبل مجمعات NET. اخرى، وتضع ملفاتها في المجلد X:\Windows\assembly، يعرف هذا المجلد بالاسم **الوحدة العامة للمجمعات Global Assembly Cache (GAC)**. يمكنك رؤية المجمعات المشتركة بجهازك من مستكشف النظام Windows Explorer وذلك بالانتقال إلى المجلد X:\Windows\assembly (**شكل 11-3 بالصفحة التالية**).



شكل 11-3: المجمعات المشتركة في وحدة GAC.

من الضروري التنبيه هنا بان مستكشف النظام Windows Explorer لايريك الشكل الفيزيـــائي الحقيقي لملفات المجمعات المشتركة، فلو تستخدم موجه الأوامـــر Command DOS Prompt، ستكتشف ان المجمعات موزعة على شكل مجلدات، كل مجلد من هذه المجلـــدات يحتـــوي علـــى مجلدات فرعية تمثل الإصدارات المختلفة للمجمع.

يمكنك حذف مجمع مشترك بالنقر بز الفأرة الأيمن على رمز المجمـــع واختيـــار الامـــر Delete من القائمة المنبثقة، اما إن اردت إضافة مجمع مشترك فتستطيع نقل ملفه إلى GAC أو استخدام الأداة GACUTIL.EXE والتي سيأتي ذكرها لاحقا في هذا الفصل.

ملاحظة

لا يمكنك حذف أو إضافة مجمع مشترك في الوحـدة العامـة للمجمعـات GAC إلا إذا كانـت لـديك صـلاحية المـستخدم Administrator مـن قبـل نظام التشغيل.

الأسماء القوية Strong Names

المجمعات الخاصة Private Assemblies يتم التفريق بينها على حسب موقع ملفاتها في القرص، لذلك إن وجدت عشرات المجمعات المتشابه في أسمائها وأرقام إصداراتها، فستنحصر المسؤولية عليك كمبرمج من وضع ملفات المجمعات في مجلدات منفصلة.

اما الحديث عن المجمعات المشتركة Shared Assemblies، فجميع المجمعات توضع فن نفس المجلد الرئيسي للوحدة GAC، وان تشابهت اسماء المجمعات، فسيتم انشاء مجلدات فرعية بارقام الإصدارات المختلفة للمجمع، فلو تخيلنا ان المجمع AAA له اصدارين، فسيتم حفظه في GAC بهذا الشكل:

```
X:\Windows\assembly\GAC\AAA\1.2.3432423
X:\Windows\assembly\GAC\AAA\2.4.12334
…
…
```

مع ذلك، الاعتماد على رقم الإصدار غير كافي وذلك لأنه قد ينشئ أكثر من مبرمج مجمع يحمل نفس الاسم ونفس رقم الإصدار وبالتالي يتم التعارض.

المفتاح العام Public Key هو الحل لمشكلة التعارض السابقة، وهو رقم كبير جدا (حجمه 128 بت) يتم إنشائه بخوارزميات معقدة بحيث تضمن عدم تعارض مفتاحين عامين في الكرة الأرضية!

النقطة الهامة التي اريد ان اصل بك هي: اسم المجمع، مفتاحه العام، رقم اصداره، اسم الشركة المنتجة، اعداداته الإقليمية ...الخ تسمى مجتمعة **بالاسم القوي للمجمع Assembly's Strong Name**.

تستطيع تسجيل اسم قوي Strong Name لمجمعاتك عن طريق الأداة SN.EXE والتي سيأتي ذكرها لاحقا.

المواصفة Assembly

يمكنك استخدام المواصفة Assembly إن اردت إسناد معلومات حول المجمع، كرقم الاصدار، اسم المجمع، الشركة، حقوق التأليف، ...الخ:

```
Imports System.Reflection
Imports System.Runtime.InteropServices
```

PDF

```
<Assembly: AssemblyTitle("نظام المحاسبة")>
<Assembly: AssemblyDescription("برنامج حسابي لادارة العمليات والحسابات
للشركات")>
<Assembly: AssemblyCompany("شبكة المطورون العرب")>
<Assembly: AssemblyCopyright("© الحقوق محفوظة لشبكة المطورون العرب")>
<Assembly: AssemblyVersion("1.0.2321432")>
…
…
…
```

بالنسبة للاصدار، فيمكنك استخدام النجمة * حتى تزيد رقم المراجعة بشكل تلقائي في كل مرة تنفذ وتجرب المجمع:

```
<Assembly: AssemblyVersion("1.0.*")>
```

اخيرا، المشاريع التي تنجزها بيئة التطوير Visual Studio .NET تقوم باضافة الشيفرات السابقة في ملف مستقل يحمل الاسم AssemblyInfo.vb، ويفضل لك إتباع نفس الأسلوب.

ملفات التهيئة Configuration Files

يمكنك التحكم أكثر في إدارة المجمعات المتعددة وتغيير معظم أساليب تنفيذها أو البحث عنها عن طريق ملفات التهيئة Configuration Files. هذه الملفات ما هي الا ملفات نصية منسقة بصيغ XML يمكنك تحريرها بأبسط برامج التحرير كالمفكرة Notepad لتغيير مجموعة من الاعدادات التي تتعلق بالمجمعات والربط بينها.

ملاحظة

ان كنت من مبرمجي Windows المخضرمين (الإصدارات 3.x وما قبلها) فالفكرة من ملفات التهيئة هي مشابهة إلى حد كبير بملفات التهيئة السابقة ذو الامتداد *.ini والتي ترفق مع التطبيقات المختلفة.

PDF Factory Pro trial version www.fineprint.com

أنواع ملفات التهيئة

في عالم NET. توجد ثلاث أنواع من ملفات التهيئة عبارة عن مستويات لتحديد الاعدادات للمجمعات NET. هي: ملف تهيئة التطبيق Application Configuration File، ملف تهيئة الناشر Publisher Configuration File، وملف تهيئة الجهاز Machine Configuration File. وفيما يلي تفاصيلها:

ملف تهيئة التطبيق Application Configuration File:

ملف التهيئة هذا يكون موجه وخاص بتطبيق أو مجمع واحد فقط، وان اردت انشاء ملف تهيئة التطبيق بنفسك، فلابد ان يكون في نفس مجلد ملف التنفيذ الرئيسي للمجمع و يحمل نفس اسم ملف التنفيذ الرئيسي مع اضافة الامتداد config. إلى امتداد الملف الرئيسي. فلو كان الاسم الكامل لملف التنفيذ الرئيسي للمجمع بهذا الشكل:

```
C:\MyProg\application.exe
```

لابد ان يكون ملف تهيئة التطبيق بهذا الاسم:

```
C:\MyProg\application.exe.config
```

ملف تهيئة الناشر Publisher Configuration File:

هذا النوع من ملفات التهيئة يستخدم مع المجمعات المشتركة Shared Assemblies بحيث يشمل تأثيره على المجمع المشترك وجميع التطبيقات التي تستخدم هذا المجمع. يمكنك الاستفادة من ملفات تهيئة الناشر ان قمت –مثلا- بتوزيع احد المجمعات المشتركة (لنقل مكتبة فئات) على المستخدمين، وبعد فترة اكتشف خطأ في هذه المكتبة وأردت من المستخدمين إعادة تثبيتها، لذلك قد ترفق ملف تهيئة من هذا النوع حتى تعيد توجيه جميع التطبيقات التي استخدمت المجمع المشترك إلى الاصدار الجديد منه.

شروط تسمية ملف تهيئة الناشر هي مثل شروط تسمية ملف تهيئة التطبيق، ولكنه موجه – كما قلت- للمجمعات المشتركة.

ملف تهيئة الجهاز Machine Configuration File:

اما ملف تهيئة الجهاز (يسمى ايضا بملف تهيئة المشرف Administrator Configuration File) يشمل تأثيره على جميع تطبيقات وبرامج .NET. التي تعمل تحت إصدار إطار عمل .NET Framework معين في الجهاز الحالي. اسم هذا الملف machine.config وتجده في المجلد C:\WINDOWS\Microsoft.NET\Framework\vxxx\CONFIG\ (حيث تمثل xxx رقم اصدار إطار عمل .NET Framework. المثبت في الجهاز)، ففي جهازي الشخصي هذا هو الاسم الكامل لملف تهيئة الجهاز:

```
C:\WINDOWS\Microsoft.NET\Framework\v1.0.3705\CONFIG\machine.config
```

تغيير الاعدادات

الشكل العام لملفات التهيئة يحمل الوسم <configuration> بصيغ XML:

```
<?xml version="1.0" encoding="UTF-8"?>
    <configuration>
        …
        …
        …
    </configuration>
```

توجد مئات الاعدادات والوسوم الإضافية التي تتحكم في سلوك تنفيذ المجمعات يمكنك إضافتها بين فكي الوسم <configuration> السابق. بعض هذه الاعدادات خاصة بملفات تهيئة التطبيقات Application configuration files فقط، وبعضها خاصة بملف تهيئة الجهاز machine.config فقط، والبعض منها يشمل جميع الأنواع المختلفة لملفات التهيئة.

في حالة تعارض الاعدادات بين ملفات التهيئة، فالأسبقية ستكون لملف تهيئة التطبيق ومن ثم ملف تهيئة الجهاز machine.config، أي ان عملية قراءة ملفات التهيئة تبدأ بملف تهيئة الجهاز ومن ثم ملف تهيئة التطبيق وأي تعارض بين الاعدادات سيطغي عليه ملف تهيئة التطبيق.

اعدادات لملفات التهيئة

احاول في الفقرات التالية والصفحات القادمة لهذا الفصل عرض بعض هذه الاعدادات، اما إن أردت معرفة جميع الاعدادات الأخرى، فلست بحاجة إلى تذكيرك بمراجع .NET Documantation:

PDF

الوسم <requiredRuntime>:

عندما تنوي تنفيذ مجمعك على جهاز اخر، فان إصدار إطار عمل NET. Framework الـذي يتطلبه هو نفس الإصدار الذي تم ترجمت البرنامج فيه. مع ذلك، تستطيع تغيير الاصدار المطلوب باستخدام الوسم <requiredRuntime> والذي يشترط وضعه في القسم الفرعي <startup> من ملف التهيئة:

```
<?xml version="1.0" encoding="UTF-8"?>
    <configuration>
        <startup>
            <requiredRuntime version="v1.0.3705" />
        </startup>
    </configuration>
```

ملاحظة

تذكر ان وسوم XML تفرق بين الحروف الصغيرة والكبيرة (أي انها case-sensitive. لذلك، اكتبها بنفس حالة الأحرف التي تراها.

إن كان رقم الاصدار المحدد في الوسم <requiredRuntime> يختلف عن رقم الإصدار الأصلي للبرنامج (أي الإصدار الذي تم ترجمة البرنامج عليه)، سيتم حفظ معلومات إضافية حول هذا الاختلاف في مسجل النظام Registry Windows، وحتى لا تكثر هذه المعلومات في كـل مجمـع تقـوم بتغييـره، يمكنـك إسـناد القيمـة True للخاصـية safemode فـي الوسـم <requiredRuntime>:

```
<?xml version="1.0" encoding="UTF-8"?>
    <configuration>
        <startup>
            <requiredRuntime version="v1.0.3705" safemode="true" />
        </startup>
    </configuration>
```

الوسم <gcConcurrent>:

تقوم المجموعة Garbage Collection بعملية تفريغ الذاكرة في مسار تنفيذ مستقل، مما يضمن عدم إيقاف مسارات التنفيذ الأخرى والتي بحاجة لعملية الاستمرار خاصة ان كانت خاصة لعرض واجهات رسومية. مع ذلك، تنصح مستندات MSDN بإلغاء هذه الطريقة وإيقـاف عمـل كافـة

مسارات التنفيذ لحظة تفريغ الذاكرة في ان كنت تنوي تطوير برامج تعمل في أجهـزة الخـوادم Servers ولا تحتوي على أية واجهات رسومية. يمكنك استخدام الوسم <gcConcurrent> فـي القسم الفرعي <runtime> لملف التهيئة وإسناد القيمة False لخاصيته enabled لوقف جميـع مسارات التنفيذ لحظة تفريغ المجموعة Garbage Collection للذاكرة:

```
<?xml version="1.0" encoding="UTF-8"?>
    <configuration>
        <runtime>
            <gcConcurrent enabled="false"/>
        </runtime>
    </configuration>
```

انظر أيضا

لمزيد من التفاصيل حول المجموعة Garbage Collection وعملية تفريغ الذاكرة، راجع الفصل الثالث **الفئات والكائنات**.

الوسم <add>:

تستطيع استخدام الوسم <appSettings> لوضع قيم خاصة بك قد يحتاجها برنامجـك (كمـسار ملفات قواعد البيانات، اعدادات واجهة الاستخدام، بيانات اخرى خاصة ببرنامجك، ...الخ)، يمكنك استخدام الوسم <add> في داخل القسم الفرعي <appSettings> لملف التهيئة:

```
<?xml version="1.0" encoding="UTF-8"?>
    <configuration>
        <appSettings>
            <add key="Site" value="www.dev4arabs.com" />
            <add key="DBPath" value="C:\Folder\Data.MDB" />
            <add key="Show Startup Window" value="False" />
        </appSettings>
    </configuration>
```

تستطيع قراءة هذه الاعدادات والخاصة بك في أي وقت من شيفراتك المـصدرية لحظـة تنفيـذ البرنامج باستخدام الفئة System.Configuration.ConfigurationSettings.AppSettings بهذا الشكل:

PDF

```
Imports System.Configuration.ConfigurationSettings

Module Module1
    Sub main()
        ' www.dev4arabs.com
        ArabicConsole.WriteLine(AppSettings("Site"))

        ' C:\Folder\Data.MDB
        ArabicConsole.WriteLine(AppSettings("DBPath"))

        ' False
        ArabicConsole.WriteLine(AppSettings("Show Startup Window"))
    End Sub
End Module
```

استخدام الأداة .NET Framework Configuration

ان كنت شخص لا يفضل تحرير ملفات التهيئة يدويا، يمكنك الاستعانة بالأداة .NET Framework Configuration (**شكل 11-4**)، والتي تصل إليها باختيار الرمز Microsoft .NET Framework Configuration من المجموعة Administrative Tools في لوحة التحكم Control Panel.

شكل 11-4: الأداة .NET Framework Configuration..

ملاحظة

الأداة .NET Framework Configuration هي برنامج يسمى في عالم Windows بـ Microsoft Management Console snap-in. وهذا النوع من البرامج لا يعمل إلا تحت الإصدارات Windows NT، Windows 2000، Windows XP، وما بعدها.

وحتى اعرض لك مثالا لاستخدامها، دعني اكرر فقرة **الوسم** <**gcConcurrent**> السابقة، يمكنك التحكم في طريقة تنفيذ المجموعة Garbage Collection في ملف تهيئة الجهاز machine.config بالضغط بزر الفأرة الأيمن على الرمز My Computer في الشجرة اليسرى، ومن ثم اختيار Properties ليظهر لك صندوق الحوار My Computer Properties (**شكل 11-4**).

شكل 11-4: تغيير سلوك تنفيذ المجموعة Garbage Collection.

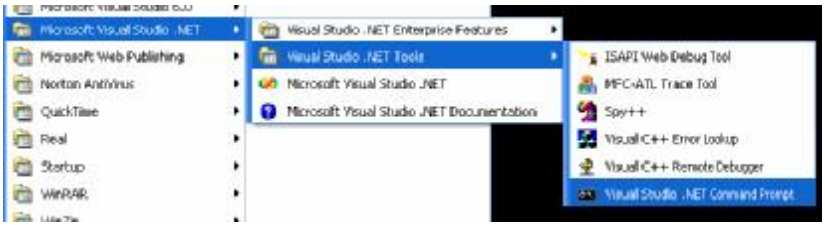
يمكنك إضافة وتصحيح ملفات تهيئة التطبيقات بالضغط على الرمز Application في الشجرة اليسرى للنافذة. وبالنسبة للمجمعات المشتركة، فستجد ضالتك بالضغط على الرمز Configured Assemblies.

أدوات الترجمة، الربط، والتسجيل

في هذا القسم اعرض لك مجموعة من الأدوات التي تستخدم لترجمة وربط الشيفرات المصدرية والوحدات المدارة والمجمعات، يمكنك استخدام بيئة التطوير Visual Studio .NET (والتي تستخدم بدورها هذه الأدوات) لإنجاز معظم المهام المطلوبة، أو استخدام هذه الأدوات مباشرة من موجه الأوامر Command Prompt.

قد تستغرب مدى الجدوى من استخدام الأدوات عن طريق موجه الأوامر Command Prompt عوضا عن بيئة التطوير Visual Studio .NET، السبب في ذلك هو ان بعض المهام لا يمكن إنجازها بالاعتماد على بيئة التطوير فقط، من أمثلة هذه المهام إنجاز مجمع متعدد الملفات، أو ترجمة شيفرات مصدرية من لغات برمجة NET. اخرى، وغيرها من الأمور.

ان قمت بتشغيل موجه الاوامر Command Prompt فلا تنسى تنفيذ الملف corvars.bat (والذي تجده في المجلد X:\Program Files\Microsoft Visual Studio .NET\FrameworkSDK\Bin) حيث يقوم بتحميل مسارات Paths هذه الأدوات ويسهل عليك الوصول لها، مع ذلك لست بحاجة إلى تنفيذ هذا الملف ان كنت قد شغلت نافذة موجه الأوامر من خلال الرمز Visual Studio .NET Command Prompt الموجود في المجموعة البرمجية Microsoft Visual Studio .NET بقائمة Start (**شكل 11-6**).



شكل 11-6: تشغيل موجه الأوامر من خلال Visual Studio .NET Command Prompt.

ملاحظة

لا اعلم لماذا لم تقم Microsoft بتوفير هذه الخدمات الإضافية من خلال بيئة التطوير .NET. Visual Studio نفسها، بدلا من الرجوع إلى أساليب ما قبل البرمجة المرئية Visual Programming واستخدام موجه الاوامر!

فيما يلي عرض سريع لهذه الأدوات، راجع مستندات .NET Documentation. لمزيد من التفاصيل حول طرق استخدامها.

المترجم VBC.EXE

المترجم VBC.EXE هو مترجم اللغة الأساسي Visual Basic Compiler والذي يقوم بترجمة الشيفرات المصدرية إلى شيفرات مكتوبة بلغة IL. أنشئ ملف (باستخدام المفكرة مثلا) باسم test.vb واكتب فيه هذه الشيفرة المبسطة:

```
' test.vb في الملف
Imports System

Module Module1
    Sub Main
        Console.WrilteLine ("Welcome")
    End Sub
End Module
```

يمكنك ترجمة الملف السابق وتحويله إلى وحدة مدارة باستخدام المترجم VBC.EXE من موجه الأوامر بهذه الصيغة:

```
C:\>vbc test.vb
```

ملاحظة

في الحقيقة الأمر السابق يؤدي إلى إنشاء مجمع أحادي الملف، ولكني استخدمت المصطلح وحدة مدارة لأبين لك انه في حال كون المجمع يحتوي على وحدة مدارة واحدة فقط، فسيصبح المصطلحين مجمع ووحدة مدارة مترادفين تقريبا.

ان كان برنامجك يحتوي على أكثر من ملف، فستستطيع إرفاقها بكتابة أسمائها جميعا:

```
C:\>vbc test.vb test2.vb test3.vb
```

سيكون اسم ملف الوحدة المدارة مثل اسم الملف المصدري الاول test.exe، وان اردت تغيير الاسم استخدم الوسيطة out/ (والتي تمكنك من استخدام النجمة * لترمز كافة الملفات):

```
C:\>vbc /out:myprog *.vb
```

الامر السابق سينشئ ملف تنفيذي باسم myprog.exe يمثل مجمع من النوع Console Application. مع ذلك، تستطيع تحديد نوع المجمع اما تطبيق Windows Application أو مكتبة Library باستخدام الوسيطة target/ (يمكنك اختصارها إلى t/):

(winprog.exe تطبيق**)**
```
C:\>vbc winprog.vb /target:winexe
```

(mylib.exe مكتبة**)**
```
C:\>vbc mylib.vb /t:library
```

استخدمنا المترجم VBC.EXE في الأمثلة السابقة لإنشاء وحدات مدارة ولكنها في نفس الوقت مضمونه ومشمولة في مجمع أحادي الملف Single File Assembly، مع ذلك تستطيع ترجمة الشيفرات إلى وحدات مدارة **دون تضمينها في مجمع** لتستفيد منها لاحقا باستخدام الرابط AL.EXE، يمكنك عمل ذلك بتحديد النوع module مع الوسيطة target/:

```
C:\>vbc test.vb /target:module
```

الأمر السابق سينشئ ملف باسم test.netmodule يمثل شيفرات الوحدة المدارة فقط (ولا يشمل المجمع)، وان كانت الوحدة المدارة تعتمد أو تستخدم انواع بيانات لوحدات مدارة اخرى، عليك اضافة هذه الوحدات المدارة باستخدام الوسيطة addmodule/:

```
C:\Test>vbc test.vb /target:module /addmodule:other.netmodule
```

وحتى ترى كيف يمكنك الاستفادة من ملفات الوحدات المدارة، استمر في القراءة وتابع الفقرة التالية **الرابط AL.EXE**.

PDF

الرابط AL.EXE

الرابط AL.EXE هو رابط المجمعات Assembly Linker، وهو الوسيلة الوحيدة التي تمكنك من انجاز مجمعات متعددة الملفات Multiple File Assemblies. الدور الرئيسي الذي يقوم به هو ربط الوحدات المدارة والمنجزة بلغات .NET. مختلفة وتكوين مجمع.

يتطلب منك الرابط AL.EXE تحديد الوحدات المدارة التي ترغب في ربطها لتكوين مجمع متعدد ملفات:

```
C:\>al file1.netmodule file2.netmodule /out:myass.dll
```

الامر السابق يقوم بإنشاء مجمع متعدد الملفات يحتوي على ثلاث ملفات file1.netmodule، file2.netmodule، و myass.dll (**شكل 11-7**). لا تنسى ان هذا المجمع مجمع متعدد الملفات أي –بعبارة أخرى- ملفات الوحدات المدارة file1.netmodule و file2.netmodule لابد ان تكون في نفس مجلد الملف myass.dll.

شكل 11-7: مجمع متعدد الملفات.

ملاحظة

حتى تواجه مستندات .NET Documentation. بمعرفة كافية، الملف myass.dll السابق هو الملف الرئيسي للمجمع ويسمى **مرشد المجمع** Assembly's Manifest.

المجمع السابق هو مجمع أسلوب تنفيذه من النوع مكتبة Library، تستطيع تكوين مجمعات من النوع Console Application أو Windows Application بإرسال الوسيطات /t:exe أو /t:win على التوالي:

```
( Console Application )
C:\>al file.netmodule /t:exe /out:myprog.exe

( Windows Application )
C:\>al file.netmodule /t:win /out:myprog.exe
```

مع ذلك، ستظهر لك رسالة خطأ مفادها عدم تحديد نقطة بداية التنفيذ Entry Point، وذلك لان المجمعات من النوع Console Application و Windows Application لابد ان يكون لها نقطة بداية. يمكنك تحديد اسم الإجراء Sub Main() الذي تود جعله نقطة البداية للمجمع بإرسال الوسيطة /main:

```
C:\test>al file.netmodule /t:exe /out:myprog.exe /main:Module1.main
```

ملاحظة

معلومة كلفتني أكثر من 4 ساعات متواصلة من التجارب الغير ناجحة لن تجدها في مستندات NET Documentation. وهي ان اسم إجراء نقطة البداية الذي تحدده مع الوسيطة /main يتعامل معه بحساسية للأحرف case-sensitive رغم ان لغة البرمجة Visual Basic .NET ليست حساسة case insensitive.

مثال تطبيقي:

والان سأريك تطبيق عملي لإنشاء مجمع متعدد الملفات يشمل وحدات مدارة مترجمة من شيفرات مصدرية بلغتين مختلفين هما Visual C# .NET و Visual Basic .NET. أنشئ ملفين file1.vb و file2.vb واكتب بهما هذه الشيفرة:

```
' file.vb في الملف
Public Module MainModule
    Public Sub Main ()
        Sub2()
    End Sub
End Module

' file2.vb في الملف
Module mdlFile2
    Sub Sub2 ()
        System.Console.WriteLine ("VB code works.")
        file3.clsfile3.Sub3()
    End Sub
End Module
```

PDF

الإجراء Sub Main() السابق يقوم باستدعاء الإجراء Sub2() والذي يعرض رسالة مفادها ان هذه شيفرة مكتوبة بلغة .NET Visual Basic، وبعد ذلك يقوم الإجراء باستدعاء الإجراء Sub3() وهو إجراء سنكتبه بلغة #C:

```
// file3.cs في الملف
namespace file3
{
    class clsfile3
    {
        static public void Sub3()
        {
            file4.clsfile4.Sub4();
        }
    }
}

// file4.cs في الملف
namespace file4
{
    class clsfile4
    {
        static public void Sub4()
        {
            System.Console.WriteLine ("C# code works.");
            System.Console.Read();

        }
    }
}
```

ملاحظة

الملفات المنجزة بلغة #C تكون امتداداتها .cs (اختصار C Sharp)، كما ان مترجم اللغة الخاص بها هو CSC.EXE.

احفظ الملفات الأربعة في مجلد C:\Test، وابدأ فورا باستخدام مترجمة لغة #C (CSC.EXE) لتترجم الملفات file3.cs و file4.csc، ولا تنسى استخدام الوسيطة target:module/ لأننا نريد وحدة مدارة دون مجمع:

```
C:\Test>csc /target:module *.cs
```

الامر السابق سينشئ ملف الوحدة المدارة file3.netmodule. والان حان دور الملفين file.vb و file2.vb وترجمتهما باستخدام المترجمة VBC.EXE، ولكن من الضروري إضافة مرجع الوحدة المدارة السابقة باستخدام addmodule/، وذلك لان الشيفرة المكتوبة في الملف file2.vb تستدعي الإجراء Sub3() والذي يتبع لوحدة مدارة خارجية:

```
C:\Test>vbc /target:module *.vb /addmodule:file3.netmodule
```

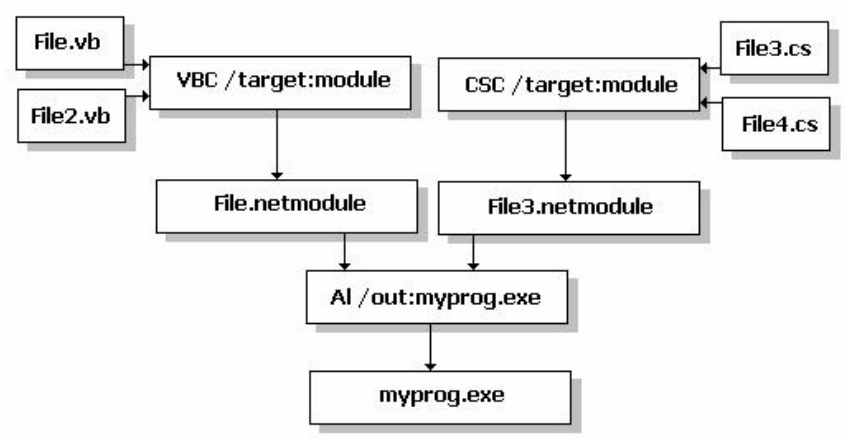
توكل على الله الذي لا تضيع ودائعه، واستخدم الرابط AL.EXE واربط كلا الوحدات المدارة لتكون مجمع من النوع Console Application (/t:exe)، نقطة بدايته الإجراء Main() (/main:MainModule.Main)، واسم ملفه الرئيسي (/out:myprog.exe):

```
C:\test>al file.netmodule file3.netmodule  /t:exe /out:myprog.exe
/main:MainModule.Main
```

مبروك! تم إنشاء المجمع المنجز بلغتي برمجة، يمكنك الان كتابة اسمه لتنفيذه كما كنا نفعل قبل عشرات السنين تحت أنظمة MS-DOS، لتكون المخرجات بهذا الشكل:

```
C:\test>myprog

VB code works.
C# code works.

C:\test> _
```

ان لم تفهم شيئا من هذا المثال، عسى ان يساعدك (**الشكل 11-8**) على استيعاب الخطوات السابقة.



شكل 11-8: مجمع متعدد الملفات File.netmodule، file3.netmodule، و myprog.exe.

المسجل SN.EXE

يستخدم المسجل SN.EXE لتسجيل اسماء قوية Strong Names لمجمعاتك، الخطوة الاولى التي تحتاجها هي إنشاء **مفتاح مركب Partial key** وحفظه في ملف منفصل (باستخدام الوسيطة k–):

```
C:\>sn -k mykey.snk
```

الامر السابق سينشئ مفتاح مركب (وهو يشمل المفتاح العام Public Key والمفتاح الخاص Private Key والذي يمثل توقيع المنتج أو الشركة).

ان كنت تود ترجمة البرنامج باستخدام المترجم VBC.EXE أو الرابط AL.EXE، عليك ارسال الملف مع المدخل keyfile/ لحظة الترجمة أو الربط:

```
C:\>vbc /out:myprog *.vb /keyfile:mykey.snk
```

اما ان كنت في داخل بيئة التطوير .NET Visual Studio، فأضف الشيفرة التالية في الملف AssemblyInfo.vb:

```
<Assembly: AssemblyKeyFile("mykey.snk")
```

المسجل GACUTIL.EXE

يمكنك تسجيل المجمع وجعله مجمعا مشتركا Shared Assembly باستخدام الأداة GACUTIL.EXE، واستخدامها سهل جدا، فكل ما تطلبه منك اسم ملف المجمع المراد تسجيله في الوحدة العامة للمجمعات GAC مع إرسال الوسيطة i/:

```
C:\>gacutil /i mylib.dll
```

يمكنك عرض جميع المجمعات في الوحدة العامة GAC بإرسال الوسيطة l/:

```
C:\>gacutil /l
```

أخيرا، تستطيع حذف مجمع مشترك من وحدة GAC بكتابة اسم المجمع **وليس اسم ملفه** مع إرفاقه بالوسيطة u/:

```
C:\>gacutil /u mylib
```

يعيب المدخل u/ السابق، انه يحذف كافة المجمعات التي تحمل الاسم mylib باختلاف إصداراتها، لذلك يفضل تحديد المجمع الذي تود حذفه بذكر إصداره (استخدم الوسيطة u– وليس u/):

```
C:\>gacutil –u mylib, ver=1.2.3432432
```

PDF

لا تعتمد على هذا الفصل ان كنت تنوي استخدام هذه الأدوات في مشاريعك الكبيرة، حيث عليك العودة إلى مكتبة MSDN وقراءة التفاصيل الدقيقة حول المجمعات والمترجمات وطريقة التعامل معها، كما عليك استخدام الأدوات السابقة بحذر شديد فهي تعتمد على نوعية الشيفرات التي تستخدمها في برامجك، اما الأمثلة السابقة فليست سوى شيفرات مبسطة هدفها توضيح الفكرة فقط.
الفصل التالي **فئات الانعكاس Reflection Classes** يمكنك من الاستعلام والحصول على معلومات حول المجمعات والوحدات المدارة التي أنشأتها.

PDF Converter

فئات الانعكاس Reflection Classes

فئات الانعكاس Reflection Classes هي مجموعة من الفئات مشمولة في مجال الاسماء System.Reflection غرضها التعامل مع المجمعات Assemblies، الوحدات المدارة Managed Modules، الفئات والكائنات Classes and Objects، الواجهات Interfaces، التركيبات سواء كانت Structures او Enums، وأعضاء البيانات (الحقول، الطرق، الخصائص، والأحداث).

في محرر الشيفرات، بعد ان تكتب نقطة "." بعد اسم الكائن، ستلاحظ ان قائمة IntelliSense تظهر لك كافة الأعضاء التابعة للكائن، كيف تمكنت بيئة التطوير Visual Studio .NET من معرفة هذه الأعضاء؟ الجواب هو ما يفصله لك هذا الفصل.

ملاحظة

بما ان فئات الانعكاس مشمولة في مجال الاسماء System.Reflection، فمن البديهي ستقوم استيراده لاختصار الشيفرات البرمجية:

```
Imports System.Reflection
```

التعامل مع المجمعات والوحدات المدارة

في هذا القسم اعرض لك فئتين تمثل المجمعات والوحدات المدارة هما: Assembly و Module. يمكنك البحث عن كافة تفاصيل خصائص وطرق هذه الفئات في مستندات .NET Documentation حيث لن تجد هنا الا عرض سريع ومختصر لأبرز أعضائها.

الفئة Assembly

الفئة Assembly تمثل –كما هو واضح من اسمها- مجمع مستقل. لا يمكنك انشاء كائن مباشرة باستخدام New من هذه الفئة، وذلك لأنك لن تنشئ مجمع جديد، بل الذي ستفعله هو العودة بمرجع للمجمع الحالي باستدعاء الطريقة المشتركة GetExecutingAssembly():

```
Dim ass As [Assembly]
ass = [Assembly].GetExecutingAssembly()
```

أو العودة بمرجع لمجمع عن طريق اسم ملفه بإرساله إلى الطريقة المشتركة LoadFrom():

```
Dim ass As [Assembly]
ass = [Assembly].LoadFrom("C:\myLib.dll")
```

أو مجموعة من الطرق الأخرى والتي تجد تفاصيلها في مكتبة MSDN.

ملاحظة

الكلمة Assembly هي كلمة محجوزة keyword للغة البرمجة Visual Basic .NET، لذلك استخدمت الأقواس [و] عند تصريح متغير من هذه الفئة:

```
Dim ass As [Assembly]
```

بعد إنشاء كائن يمثل مجمع، تستطيع الاستعلام عن مجموعة كبيرة من محتوياته، كاسم المجمع الكامل (يشمل الاسم، رقم الاصدار، الاعدادات الإقليمية، والمفتاح العام Public Key) عن طريق الخاصية FullName، كما يمكنك معرفة ما اذا كان المجمع قادم من الوحدة العامة للمجمعات GAC عن طريق الخاصية GlobalAssemblyCache، والخاصية Location التي تعود بمسار ملف المجمع:

```
Dim ass As [Assembly]
ass = [Assembly].GetExecutingAssembly
ArabicConsole.WriteLine(ass.FullName)
ArabicConsole.WriteLine(ass.GlobalAssemblyCache)     ' False
ArabicConsole.WriteLine(ass.Location)        ' C:\test.dll
```

PDF

من الخصائص الاخرى، الخاصية EntryPoint والتي تعود بكائن من النوع Refelection.MethodInfo تمثل إجراء نقطة البداية للمجمع:

```
Dim ass As [Assembly]
Dim mthd As MethodInfo

ass = [Assembly].GetExecutingAssembly
mthd = ass.EntryPoint

ArabicConsole.WriteLine(mthd.Name) ' main
```

ستعود الخاصية EntryPoint السابقة بالقيمة Nothing ان كان المجمع من النوع مكتبة Library، وذلك لان هذا النوع من المجمعات لا يحتوي على نقطة بداية.

انظر أيضا

راجع الفصل السابق **المجمعات** Assemblies لمعرفة ماذا يقصد بنقطة البداية Entry Point، وبالنسبة للفئة MethodInfo فسأتحدث عنها لاحقا في هذا الفصل.

الطريقة GetTypes() تعود بمصفوفة من النوع Type تمثل جميع البيانات المعرفة (الفئات، الواجهات، التركيبات، ...الخ) في المجمع:

```
Dim ass As [Assembly]
Dim t As Type
ass = [Assembly].LoadFrom("C:\myLib.dll")

For Each t In ass.GetTypes()
    ArabicConsole.WriteLine(t.Name)
Next
```

الطريقة GetTypes() السابقة تعود بجميع الأنواع المعرفة في المجمع (بما فيها التي لا تستطيع الوصول لها كالمعرفة بمحدد الوصول Friend أو Private)، لذلك يفضل استخدام الطريقة GetExportedTypes() والتي تعود بالأنواع المعرفة بمحدد الوصول Public والتي يمكنك الوصول لها:

```
Dim ass As [Assembly]
Dim t As Type
ass = [Assembly].LoadFrom("C:\myLib.dll")

For Each t In ass.GetExportedTypes()
    ArabicConsole.WriteLine(t.Name)
Next
```

اخيرا، الربط المتأخر Late Binding والذي تعرفنا عليه في الفصول الأولى من هذا الكتاب يستخدم فئات الانعكاس أيضا لإنجازه، لديك الطريقة CreateInstance() لتتمكن من إنشاء كائن بالربط المتأخر وذلك بارسال اسمه البرمجي:

```
Dim ass As [Assembly]
ass = [Assembly].LoadFrom("C:\test\test.dll")
Dim obj As Object = ass.CreateInstance("LibNamespace.PublicClass")

' اثبات انه تم انشاء الكائن
ArabicConsole.WriteLine(obj.GetType.FullName) '
LibNamespace.PublicClass
```

الفئة Module

اما الفئة Module فهي تمثل الوحدات المدارة Managed Modules والتابعة للمجمع، يمكنك إنشاء كائن من هذه الوحدة باستدعاء الطريقة GetModule() التابعة للفئة Assembly، أو الطريقة GetModules() والتي تعود بمصفوفة تمثل كافة الوحدات المدارة في المجمع:

```
Dim ass As [Assembly]
ass = [Assembly].LoadFrom("C:\test\test.dll")

Dim mdl As [Module]

For Each mdl In ass.GetModules
    ArabicConsole.WriteLine(mdl.Name)
Next
```

وكما هو الحال مع الفئة Assembly، تحتوي الفئة Module على الطريقة GetTypes() ولكنها تعود بجميع الانواع المعرفة في الوحدة المدارة وليس كامل المجمع.

التعامل مع أنواع البيانات

في هذا القسم اعرض لك المزيد من الطرق والخصائص التي تمكنك من الاستعلام عن الطرق والخصائص وكافة أعضاء البيانات التي تريدها لحظة التنفيذ. ولكن قبل ذلك، احتاج إلى تعريفك اكثر بالفئة System.Type والتي اذكر أننا استخدمناها سابقا في مواقع متفرقة من هذا الكتاب.

الفئة System.Type

جميع فئات الانعكاس –كما ذكرت- مشمولة في مجال الاسماء System. Reflection ما عدا الفئة Type فهي في مجال الاسماء الجذري System. مع ذلك، تصنف الفئة System.Type من فئات الانعكاس ايضا، فهي تمثل نوع معين من البيانات، قد يكون هذا النوع أولي Primitive Type (كـ Integer، Long، String، Boolean ...الخ)، أو نوع معرف كفئة Class، تركيب Strucuter أو Enum، واجهة Interface.

لن تنشئ كان جديد من هذه الفئة باستخدام New مباشرة، وانما ستحاول اسناد مرجع إلى الكائنات من النوع Type بأساليب عديدة، منها –مثلا- باستدعاء الطريقة GetTypes() والتابعة للفئة Assembly السابقة:

```
Dim ass As [Assembly]
Dim t As Type
ass = [Assembly].LoadFrom("C:\myLib.dll")

For Each t In ass.GetTypes()
    ArabicConsole.WriteLine(t.Name)
Next
```

أو تسهل علي نفسك المهمة اكثر باستخدام الدالة GetType() والتابعة للغة البرمجة Visual Basic .NET:

```
Dim t As Type

t = GetType(Integer)
ArabicConsole.WriteLine(t.FullName) ' System.Int32
```

لو لم تخونك الذاكرة فستذكر في بداية الفصل السادس **الفئات الأساسية** ان الفئة System.Object (والتي ترث منها جميع فئات إطار عمل Framework .NET. الأخرى)

PDF

تحتوي على الطريقة ()GetType والتي تعود بكائن من النوع Type يمثل نوع الكائن الحالي حتى لو كان الكائن منشئ بالربط المتأخر Late Binding:

```
Dim t As Type
Dim obj As Object = New TestClass()

t = obj.GetType()

ArabicConsole.WriteLine(t.FullName) ' MyNamespace.TestClass
```

المزيد ايضا، الطريقة ()GetType السابقة تم إعادة تعريفها Overloads في الفئة System.Type بحيث تمكنك من ارسال قيمة حرفية من النوع Sring:

```
Dim t As Type
t = Type.GetType("System.Double")

Console.WriteLine(t.FullName)
```

دعني الفت انتباهك بان الاسم الحرفي الذي ترسله للطريقة حساس لحالة الاحرف Case-Sensetive، كما عليك استخدام الاسماء الأصلية لإطار عمل .NET Framework. (كـ Int32، Int64، DateTime ....الخ) عوضا عن الأسماء الخاصة بلغة البرمجة .NET Visual Basic (كـ Integer، Long، Date ...الخ).

بالإضافة إلى الطريقة ()GetType، لديك الطريقة ()GetTypeArray والتي ترسل معها مصفوفة لكائنات، لتعود بنوع كل كائن من هذه المصفوفة على حده:

```
Dim X() As Object = {"عباس", 100, 3.5}
Dim t() As Type = Type.GetTypeArray(X)


ArabicConsole.WriteLine(t(0).FullName) ' System.String
ArabicConsole.WriteLine(t(1).FullName) ' System.Int32
ArabicConsole.WriteLine(t(2).FullName) ' System.Double
```

خاص لمبرمجي المكونات COM:

ان كنت من مبرمجي المكونات COM المخضرمين، فدعني اهمس في اذنك وأخبرك ان الفئة Type يمكن لها ان تحمل كائناتها انواع بيانات لفئات منجزة بتقنية COM، تستطيع عمل ذلك باستدعاء الطريقة ()GetTypeFromProgID وارسال قيمة **معرف ProgID** (برمجة المكونات COM موضوع أقدم من ان يذكر في هذا الكتاب):

PDF Factory Pro trial version www.fastio.com

```
' Microsoft Word استخدام احد فئات
Dim word As Type = Type.GetTypeFromProgID("Word.Application")

ArabicConsole.WriteLine(word.FullName) ' System.__ComObject
```

خصائص إضافية

توجد عشرات الخصائص للقراءة فقط التي تمكنك من الحصول على تفاصيل اكثر للنوع المسند إلى الكائن من النوع Type، من هذه الخصائص الخاصية Name التي تعود بالاسم البرمجي للنوع، الخاصية FullName التي تشمل مجال الاسماء في الاسم البرمجي للنوع، الخاصية Module التي تعود بالوحدة المدارة Managed Module التي عرف بها النوع، والخاصية Assembly التي تعود بالمجمع الذي عرف في النوع:

```
Dim t As Type = GetType(String)

ArabicConsole.WriteLine(t.Name)        ' String
ArabicConsole.WriteLine(t.FullName)    ' System.String
ArabicConsole.WriteLine(t.Module)      ' CommonLanguageRuntimeLibrary
```

خصائص منطقية أخرى تعود بقيم من النوع Boolean هي: IsClass، IsInterface، IsEnum، IsValueType والتي تكتشف فيها ما اذا كان النوع يمثل فئة Class، واجهة Interface، تركيب من النوع Enum، نوع ذات قيمة Value Type أو تركيب من النوع Structure على التوالي:

```
Dim t As Type = GetType(TestClass)

ArabicConsole.WriteLine(t.IsClass)          ' True
ArabicConsole.WriteLine(t.IsValueType)      ' False
```

يمكنك معرفة ايضا محدد الوصول الذي أستخدم لتعريف النوع عن طريق الخاصية IsPublic التي تعود بالقيمة True ان كان محدد الوصول Public، الخاصية IsNestedPrivate التي تعود بالقيمة True ان كان محدد الوصول Private، والخاصية IsNestedAssembly التي تعود بالقيمة True ان كان محدد الوصول Friend، الخاصية IsNestedFamily التي تعود بالقيمة True ان كان محدد الوصول Protected، والخاصية IsNestedFamORAssem التي تعود بالقيمة True ان كان محدد الوصول Protected Friend.

وبالنسبة للعلاقات الوراثية بين الفئات، فيمكنك معرفة ما اذا كان النوع غير قابل للوراثة NotInheritable ان كانت قيمة الخاصية IsSealed هي True، وعلى العكس تعود الخاصية IsAbstract بالقيمة True ان كان النوع MustInherit.

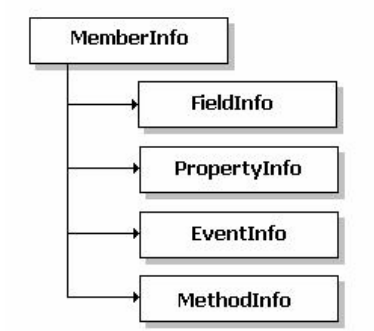
نصيحتي لك بان تراجع مكتبة MSDN ان اردت معرفة اكبر قدر من الخصائص.

التعامل مع الأعضاء

في القسم السابق عرضت لك طرق وخصائص تمكنك من الاستعلام عن انواع البيانات، وفي هذا القسم سنفصل في هذه الاستعلامات بحيث تعرض لك أعضاء (حقول، خصائص، طرق، وأحداث) النوع. أعيد واكرر نصيحتي السابقة (التي أحس بانها أصبحت مملة) في العودة إلى مراجع MSDN للحصول على كافة الطرق والخصائص.

الفئة القاعدية MemberInfo

الفئة MemberInfo تمثل عضو Member سواء كان (حقل، خاصية، حدث، أو طريقة) في النوع، وهي بدورها تعتبر الفئة القاعدية لاربع فئات اخرى هي: FieldInfo، PropertyInfo، EventInfo، و MethodInfo (**شكل 12-1**).



شكل 12-1: الفئة القاعدية MemberInfo

حتى تتمكن من استيعاب الفئات الاربعة المشتقة، سيسهل عليك كثير إن أتقنت الفئة القاعدية MemberInfo، ولكي تتعامل مع كائنات الفئة القاعدية MemberInfo، فلن تتمكن من إنشائها باستخدام New وانما عليك الحصول إلى مرجع العضو في النوع، يتم ذلك باستدعاء الطريقة GetMembers() والتي بكافة الأعضاء التابعة للنوع:

```
Dim T As Type = GetType(Integer)
Dim m As MemberInfo

For Each m In T.GetMembers
    ArabicConsole.WriteLine(m.Name)
Next
```

مخرجات الشيفرة السابقة ستكون أعضاء النوع Integer:

```
MaxValue
MinValue
ToString
GetTypeCode
ToString
CompareTo
GetHashCode
Equals
ToString
ToString
Parse
Parse
Parse
Parse
GetType
```

ان ركزت في المخرجات السابقة، ستلاحظ ان بعض الأعضاء قد تكررت، والسبب في ذلك ان هذه الأعضاء تابعة لفئات مشتقة أو تم إعادة قيادتها Overrides أكثر من مرة، لذلك حتى تتجنب التكرار تستطيع إرسال مجموعة من القيم إلى الطريقة GetMembers() ما هي إلا تركيب من النوع Enum باسم BindingFlags، والذي يمكنك من حصر الأعضاء على عدة أوجه، منها:

- القيمة BindingFlags.Public تحصر المجال للأعضاء العامة، والقيمة BindingFlags.NonPublic للأعضاء الغير عامة.
- القيمة BindingFlags.Static تعود بالأعضاء المشتركة Shared Members، والقيمة BindingFlags.Instance للأعضاء التقليدية Instance Members.

PDF

§ القيمة BindingFlags.DeclaredOnly تعود بالأعضاء المعرفة في النوع الحالي فقط –اي لا تشمل أعضاء الفئات القاعدية.

فمثلا، الشيفرة التالية ستعود بالأعضاء التقليدية Instance Members، على المستوى العام Public، وتشمل الأعضاء المصرحة في النوع ومتجاهلة أعضاء الفئات القاعدية:

```
Dim T As Type = GetType(Integer)
Dim m As MemberInfo

For Each m In T.GetMembers(BindingFlags.Public _
    Or BindingFlags.Instance Or _
    BindingFlags.DeclaredOnly)

    ArabicConsole.WriteLine(m.Name)
Next
```

مخرجات الشيفرة السابقة ستكون شيئا مثل:

```
ToString
GetTypeCode
ToString
CompareTo
GetHashCode
Equals
ToString
ToString
```

عليك معرفة ان تكرار بعض الأعضاء في المخرجات السابق ناتج عن كونها معاد تعريفها Overloads في نفس النوع، ولا تتبع للفائت القاعدية.

الطرق والخصائص:

تحتوي الفئة MemberInfo على عشرات الطرق والخصائص التي تمكنك من الاستعلام حول الأعضاء والخاصية بالنوع المرسل، لديك مثلا الخاصية Name التي تعود باسم العضو، الخاصية MemberType التي تعود بنوع العضو (حقل Field، طريقة Method، خاصية Property ...الخ)، والخاصية Attributes التي تعود بالمواصفة Attribute المستخدمة مع العضو.

من الخصائص التي تمكنك من معرفة محدد الوصول المستخدم لتعريف العضو الخاصية IsPublic التي تعود بالقيمة True ان كان محدد الوصول Public، الخاصية IsPrivate التي

تعود بالقيمة True ان كان محدد الوصول Private، والخاصية IsAssembly التي تعود بالقيمة True ان كان محدد الوصول Friend، الخاصية IsFamily التي تعود بالقيمة True ان كان محدد الوصول Protected، والخاصية IsFamilyORAssembly التي تعود بالقيمة True ان كان محدد الوصول Protected Friend.

اخيرا، يمكنك معرفة ما اذا كان العضو مشترك Shared أو لا عن طريق الخاصية IsStatic.

التعامل مع الحقول

الفئة FieldInfo تمثل حقل Field تابع لنوع بيانات، تحتوي الفئة FieldInfo على مجموعة من الخصائص والطرق الخاصة بها، أو المشتقة من الفئة القاعدية MemberInfo. ولكي تتعامل مع كائنات الفئة FieldInfo، فلن تتمكن من إنشائها باستخدام New وإنما عليك الحصول إلى مرجع العضو في النوع، يتم ذلك باستدعاء الطريقة GetFields() والتي تعود بكافة الحقول التابعة للنوع (بافتراض ان لدينا الفئة Person التي تحتوي على اربعة حقول):

```
Class Person
    Public Name As String
    Public BirthDate As Date
    Public Salary As Decimal
    Public Address As String
End Class
…
…
…
Dim T As Type = GetType(Person)
Dim f As FieldInfo

For Each f In T.GetFields()
    ArabicConsole.WriteLine(f.Name)
Next
```

مخرجات الشيفرة السابقة ستكون:

```
Name
BirthDate
Salary
Address
```

PDF

ملاحظة

الطريقة GetFields() تستقبل وسيطة من النوع BindingFlags وهي تماثل وسيطة الطريقة GetMembers().

الطرق والخصائص:

من خصائص الفئة FieldInfo الخاصية IsNotSerialized التي تعود بالقيمة False ان كان الحقل قابل للتسلسل Serializable، والخاصية FieldType التي تعود بقيمة من النوع Type تمثل نوع الحقل:

```
Dim T As Type = GetType(Person)
Dim F As FieldInfo = T.GetField("BirthDate")

ArabicConsole.WriteLine(F.FieldType.ToString) ' System.DateTime
```

وللحديث عن الطرق، فالفئة FieldInfo تحتوي على الطرقتين SetValue() و GetValue() اللتان تمكنانك من إسناد أو قراءة قيمة إلى الحقل بأسلوب الربط المتأخر Late Binding.

التعامل مع الخصائص

الفئة PropertyInfo تمثل خاصية Property تابعة لنوع بيانات، تحتوي الفئة PropertyInfo على مجموعة من الخصائص والطرق الخاصة بها، أو المشتقة من الفئة القاعدية MemberInfo. ولكي تتعامل مع كائنات الفئة PropertyInfo، فلن تتمكن من إنشائها باستخدام New وإنما عليك الحصول إلى مرجع العضو في النوع، يتم ذلك باستدعاء الطريقة GetProperties() والتي تعود بكافة الخصائص التابعة للنوع:

```
Dim T As Type = GetType(String)
Dim P As PropertyInfo


For Each P In T.GetProperties()
    ArabicConsole.WriteLine(P.Name)
Next
```

مخرجات الشيفرة السابقة ستكون:

```
Chars
Length
```

ملاحظة

الطريقة GetProperties() تستقبل وسيطة من النوع BindingFlags وهي تماثل وسيطة الطريقة GetMembers().

الطرق والخصائص:

من خصائص الفئة PropertyInfo الخاصية CanRead التي تعود بالقيمة True ان كانت الخاصية قابلة للقراءة، والخاصية CanWrite ان كانت قابلة للكتابة، كما تحتوي على الخاصية PropertyType التي تعود بقيمة من النوع Type تمثل نوع الخاصية:

```
Dim T As Type = GetType(String)
Dim P As PropertyInfo = P.GetProperty("Length")

ArabicConsole.WriteLine(P.PropertyType.ToString) ' System.Int32
```

والحديث عن الطرق، فالفئة FieldInfo تحتوي ايضا على الطريقتين SetValue() و GetValue() اللتان تمكناك من إسناد أو قراءة قيمة إلى الخاصية بأسلوب الربط المتأخر Late Binding. والطريقتين GetGetMethod() و GetSetMethod() اللتان تعودان بكائن من النوع MethodInfo يمثلنا اجراء استرجاع القيمة (Property Get) وإسناد القيمة للخاصية (Property Set) –فكما تعلم ان الخصائص تحتوي على اجرائين:

```
Property xxxx() As yyyy
    Get
        ' اجراء استرجاع القيمة
    End Get

    Set(ByVal Value As yyyy)
        ' اجراء اسناد القيمة
    End Set
End Property
```

PDF

التعامل مع الطرق

الفئة MethodInfo تمثل طريقة Method تابعة لنوع بيانات، تحتوي الفئة على مجموعة من الخصائص والطرق الخاصة بها، أو المشتقة من الفئة القاعدية MemberInfo. ولكي تتعامل مع كائنات الفئة MethodInfo، فلن تتمكن من إنشائها باستخدام New وانما عليك الحصول على مرجع العضو في النوع، يتم ذلك باستدعاء الطريقة GetMethods() والتي تعود بكافة الطرق التابعة للنوع:

```
Dim T As Type = GetType(Double)
Dim M As MethodInfo

For Each M In T.GetMethods
    ArabicConsole.WriteLine(M.Name)
Next
```

مخرجات الشيفرة السابقة ستكون:

```
ToString
GetTypeCode
ToString
CompareTo
GetHashCode
Equals
ToString
IsInfinity
IsPositiveInfinity
IsNegativeInfinity
IsNaN
ToString
Parse
Parse
Parse
Parse
TryParse
GetType
```

ملاحظة

الطريقة GetMethods() تستقبل وسيطة من النوع BindingFlags وهي تماثل وسيطة الطريقة GetMembers().

PDF

الطرق والخصائص:

من خصائص الفئة MethodInfo الخاصية ReturnType التي تعود بقيمة من النوع Type تمثل نوع القيمة التي تعود بها الطريقة (إن كانت Function بكل تأكيد)، والخاصية IsAbstract التي تعود بالقيمة True ان كانت الطريقة مصرحة على انها MustOverride، والخاصية IsVirtual تعود ايضا بالقيمة True ان كانت الطريقة مصرحة بـ Overridable.

وللحديث عن الطرق، فالفئة MethodInfo تحتوي على الطريقة ()Invoke والتي تقوم بتنفيذ الطريقة من خلال الربط المتأخر Late Binding.

ملاحظة

معظم خصائص وطرق الفئة MethodInfo مدعومة أيضا في الفئة PropertyInfo السابقة والعكس صحيح.

التعامل مع الأحداث

الفئة EventInfo تمثل حدث Event تابع لفئة، تحتوي الفئة EventInfo على مجموعة من الخصائص والطرق الخاصة بها، أو المشتقة من الفئة القاعدية MemberInfo. ولكي تتعامل مع كائنات الفئة EventInfo، فلن تتمكن من انشائها باستخدام New وانما عليك الحصول على مرجع العضو في النوع، يتم ذلك باستدعاء الطريقة ()GetEvents والتي تعود بكافة الأحداث التابعة للنوع (بافتراض ان لدينا الفئة TestClass التي تحتوي على ثلاثة أحداث):

```
Class TestClass
    Event XXX()
    Event YYY()
    Event ZZZ()
End Class
…
…
…
Dim T As Type = GetType(TestClass)
Dim E As EventInfo

For Each E In T.GetEvents
    ArabicConsole.WriteLine(E.Name)
Next
```

مخرجات الشيفرة السابقة ستكون:

```
XXX
YYY
ZZZ
```

ملاحظة

الطريقة GetEvents() تستقبل وسيطة من النوع BindingFlags وهي تماثل وسيطة الطريقة GetMembers().

الطرق والخصائص:

لا تحتوي الفئة EventInfo على خصائص إضافية خاصة بها غير المشتقة من الفئة MemberInfo.

اما الحديث عن الطرق، فالفئة EventInfo تحتوي على الطريقة GetAddMethod() التي تعود بكائن من النوع MethodInfo تمثل الاجراء الذي قام بقنص حدث الكائن باستخدام AddHandler()، بينما الطريقة GetRemoveMethod() تعود بالإجراء الذي أوقف عملية القنص باستخدام RemoveHandler(). اما الطريقة GetRaiseMethod() فتعود بكائن من النوع MethodInfo ايضا ولكنه يمثل الإجراء الذي يقنص الحدث (اي الإجراء الذي سيتم تنفيذه لحظة وقوع الحدث).

الوسيطات Parameters

الفئات من النوع MethodInfo و PropertyInfo تحتوي على الخاصية GetParameters والتي تعود بمصفوفة من النوع ParameterInfo تمثل وسيطة Parameter يستقبلها الإجراء. تحتوي الفئة ParameterInfo على مجموعة من الطرق والخصائص والخاصة بهذه الوسيطات، كالخاصية Name التي تعود باسم الوسيطة، والخاصية ParameterType التي تعود بنوع الوسيطة.

ان عادت الخاصية IsOptional بالقيمة True فهذا يعني ان الوسيطة اختيارية Optional، ويمكنك معرفة القيمة الافتراضية للوسيطة الاختيارية عن طريق الخاصية DefaultValue.

PDF

الشيفرة التالية توضح طريقة الاستعلام عن وسيطات الطريقة MySub() والتابعة للفئة TestClass:

```
Module MainModule

    Class TestClass
        Sub MySub(ByVal x As Integer, ByRef y As String)

        End Sub
    End Class

    Sub main()
        Dim T As Type = GetType(TestClass)
        Dim P As ParameterInfo

        For Each P In T.GetMethod("MySub").GetParameters()
            ArabicConsole.WriteLine(P.Name & " As " & _
                P.ParameterType.ToString)
        Next

    End Sub
End Module
```

مخرجات الشيفرة السابقة ستكون شيئا مثل:

```
x As System.Int32
y As System.String
```

التعامل مع الكائنات

في الأقسام السابقة، تعاملنا مع انواع البيانات كـ String، Integer، Double، Person، TestClass ...الخ، وكان كل ما فعلناه هو الاستعلام عنها والحصول على معلومات حولها. اما في هذا القسم سنحاول الاستفادة من الانعكاس Reflection ونتعامل مع الكائنات، لنتمكن –مثلا- من اسناد قيمة لخاصية أو استدعاء طريقة.

الفئة ReflectionExample

في هذا القسم سنجري مجموعة كبيرة من التجارب للقراءة والكتابة إلى حقول وخصائص الكائن، بالاضافة إلى استدعاء طرقه وإرسال وسيطات لها، لذلك فضلت عرض الفئة ReflectionExample والتي سنجري عليها كافة التجارب –كما بالصفحة التالية:

```
Class ReflectionExample
    Public ExampleField As String

    Private m_ExampleProperty As Integer
    Public Property ExampleProperty() As Integer
        Get
            Return m_ExampleProperty
        End Get
        Set(ByVal Value As Integer)
            m_ExampleProperty = Value
        End Set
    End Property

    Public Sub ExampleMethod(ByVal x As Integer, ByVal y As Integer)
        ArabicConsole.WriteLine(-x)
        ArabicConsole.WriteLine(-y)
    End Sub

    Public Sub ExampleMethod2(Optional ByVal y As Integer = -1)
        ArabicConsole.WriteLine(y)
    End Sub
End Class
```

أحب ان انوه هنا بان الأمثلة في الفقرات التالية مباشرة لا تستخدم اي طرق ووسائل إضافية للتحقق من الأعضاء (كالتحقق من قابلية القراءة والكتابة، محددات الوصـــول، ...الـــخ)، وذلـــك لاختصار وتسهيل الشيفرات البرمجية عليك –وعلي أنا ايضا!

إسناد/ قراءة قيم الحقول

حاول الحصول على مرجع الحقل باســتدعاء الطريقـــة GetFields() أو GetField() أو بـــأي طريقة اخرى، واسند القيمة إلى كائن من النوع FieldInfo لتستدعي طريقته GetValue() والتي تعود بقيمة تمثل قيمة الحقل:

```
Dim obj As New ReflectionExample()
Dim F As FieldInfo = obj.GetType.GetField("ExampleField")

obj.ExampleField = "***"

' التحقق من ان الحقل تم قرائة بشكل صحيح
ArabicConsole.WriteLine(F.GetValue(obj)) ' ***
```

في المقابل، استدعي الطريقة ()SetValue لاسناد قيمة للحقل، تتطلب هذه الطريقة وسيطة اضافية تمثل القيمة المراد اسنادها للحقل:

```
Dim obj As New ReflectionExample()
Dim F As FieldInfo = obj.GetType.GetField("ExampleField")

F.SetValue(obj, "$$$")

' التحقق من نجاح عملية اسناد القيمة
ArabicConsole.WriteLine(obj.ExampleField)
```

إسناد/قراءة قيم الخصائص

حاول الحصول على مرجع الخاصية باستدعاء الطريقة ()GetProperties أو ()GetProperty أو باي طريقة اخرى، واسند القيمة إلى كائن من النوع PropertyInfo لتستدعي طريقته ()GetValue والتي تعود بقيمة تمثل قيمة الخاصية، من الضروري ارسال القيمة Nothing كوسيطة ثانية لهذه الطريقة، وذلك لان هذه الخاصية لا تحتوي على أية وسيطات نرسلها اليها:

```
Dim obj As New ReflectionExample()
Dim P As PropertyInfo = obj.GetType.GetProperty("ExampleProperty")

obj.ExampleProperty = 111
' التحقق من ان الخاصية تم قرائتها بشكل صحيح
ArabicConsole.WriteLine(P.GetValue(obj, Nothing)) ' 111
```

في الفقرة التالية **استدعاء** الطرق سأريك مثالا لارسال وسيطات إلى الإجراء، اما الان فلنكمل عملنا مع الخاصية ExampleProperty1 ونحاول إسناد قيمة لها بالطريقة ()SetValue (لا تنسى ارسال القيمة Nothing مع الوسيطة الثانية):

```
Dim obj As New ReflectionExample()
Dim P As PropertyInfo = obj.GetType.GetProperty("ExampleProperty")

P.SetValue(obj, 999, Nothing)
' التحقق من نجاح عملية اسناد القيمة
ArabicConsole.WriteLine(obj.ExampleProperty) ' 999
```

استدعاء الطرق

بعد حصولك على مرجع للطريقة وإسنادها في كائن من النوع MethodInfo، يمكنك في اي وقت استدعائها بالطريقة ()Invoke والتي تتطلب وسيطات الطريقة، ان كانت الطريقة لا تحتوي على

أية وسيطات فأرسل القيمة Nothing، اما إن وجدت فأرسل مصفوفة من النـــوع Object عـــدد عناصرها يماثل عدد وسيطات الطريقة المستدعاة:

```
Dim obj As New ReflectionExample()
Dim M As MethodInfo = obj.GetType.GetMethod("ExampleMethod")
Dim params() As Object = {-5, 5}

M.Invoke(obj, params)
```

مخرجات الشيفرة السابقة ستكون:

```
5
-5
```

ان كـــان للطريقـــة وســـيطات اختياريـــة Optional Parameters، فأرســـل القيمـــة Type.Missing في المصفوفة المراد إرسالها كقيم لوسيطات الطريقة، في الشيفرة التالية قمـــت بعمل استدعائين للطريقة ()ExampleMethod2، الأول لا يرسل قيمة للوســـيطة الاختياريـــة، والثاني يرسل القيمة 10:

```
Dim obj As New ReflectionExample()
Dim M As MethodInfo = obj.GetType.GetMethod("ExampleMethod2")
Dim params() As Object = {Type.Missing}
Dim params2() As Object = {10}

M.Invoke(obj, params)
M.Invoke(obj, params2)
```

لتكون المخرجات:

```
-1
10
```

مواضيع أخرى

في هذا القسم اختم الفصل بعرض طريقة الإنشاء الديناميكي للكائنات لحظة التنفيذ (كما يفعل الربط المتأخر Late Binding، وطريقة معرفة الإجراءات المستدعية.

الإنشاء الديناميكي للكائنات

في الأقسام السابقة، تمكنا من استدعاء طرق وخصائص الكائن، ولكن هذا الكائن منشأ من شيفراتنا المصدرية لحظة التصميم:

```
Dim obj As New ReflectionExample()
```

مع ذلك، فالفائدة الحقيقية التي تجنيها من الانعكاس Reflection هي امكانية انشاء الكائنات لحظة التنفيذ بذكر الاسم النصي للكائن والذي تكون صيغته مبتدئة باسم المجمع ثم النوع AssemblyName.TypeName. يمكنك عمل ذلك، باستدعاء الطريقة CreateInstance() التابعة للفئة System.Activator والتي تتطلب وسيطة من النوع Type تمثل النوع:

```
Dim T As Type = Type.GetType("MyAssembly.MyClass")
Dim obj As Object = Activator.CreateInstance(T)
```

ان كان النوع يحتوي على مشيد Constructor فعليك إرسال وسيطات ذلك المشيد مع الطريقة CreateInstance() على شكل مصفوفة من النوع Object:

```
Dim T As Type = Type.GetType("MyAssembly.MyClass")
Dim params() As Object = {-5, 5}
Dim obj As Object = Activator.CreateInstance(T, params)
```

طريقة اخرى تمكنك من إنشاء الكائن ديناميكيا دون استخدام الفئة System.Activator، ولكنها طويلة بعض الشيء تتطلب التوغل في تفاصيل مشيد الفئة لإرساله إلى كائن من النوع ConstructorInfo (وهو مشتق من الفئة MemberInfo):

```
Dim T As Type = Type.GetType("MyAssembly.MyClass")

Dim types() As Type = { GetType(Integer), GetType(Integer) }

Dim con As ConstructorInfo = T.GetConstructor (types)

Dim params() As Object = {-5, 5}

Dim obj As Object = con.Invoke(params) ' إنشاء الكائن
```

PDF

معرفة الإجراءات المستدعية

في الفصل السابع **اكتشاف الأخطاء** علمنا ان الخاصية StackTrace التابعة لكائن الاستثناء Exception تعود بقيمة حرفية تمثل طابور الإجراءات المتعلقة بالاستثناء، كيف تمكنك هذه الفئة من معرفة ذلك؟ والجواب بفضل مجموعة من الفئات في مجال الأسماء System.Diagnostics –تصنف من فئات الانعكاس Reflection ايضا.

تمثل الفئة StackTrace الإجراءات المعرفة في طابور الاستدعاءات، تشمل الإجراءات في هذا الطابور الإجراءات المستدعية في قائمة الانتظار ليعود التنفيذ لها والمستدعاة التي يجري تنفيذها حاليا، يمكنك إنشاء كائن من الفئة StackTrace باستخدام New، وعليك استدعاء الطريقة GetFrame() التي تعود بكائن من النوع StackFrame يمثل القسم المستدعي، الكائن StackFrame بدوره يتطلب منك استدعاء طريقته GetMethod() الذي يعود بكائن من النوع MethodBase (يتبع إلى مجال الاسماء System.Collection واستخدامه شبيه بـ MethodInfo)، الشيفرة التالية تعود بكافة الإجراءات في طابور الاستدعاءات Stack Trace:

```
Dim ST As New StackTrace()
Dim counter As Integer

For counter = 0 To ST.FrameCount - 1
    Dim SF As StackFrame = ST.GetFrame(counter)
    Console.WriteLine(SF.GetMethod.Name & " ()")
Next
```

ارفق لك هذا المثال التطبيقي الذي يظهر لك طابور الاستدعاءات من الاجرائين CallerSub() و CalleeSub():

```
Imports System.Reflection
Imports System.Diagnostics

Module Module1
    Sub main()
        CallerSub()
    End Sub

    Sub CallerSub()
        Dim ST As New StackTrace()
        Dim counter As Integer

        ArabicConsole.WriteLine("** CallerSub *********")
```

```
        For counter = 0 To ST.FrameCount - 1
            Dim SF As StackFrame = ST.GetFrame(counter)
            ArabicConsole.WriteLine(SF.GetMethod.Name & " ()")
        Next

        CalleeSub()
    End Sub

    Sub CalleeSub()
        Dim ST As New StackTrace()
        Dim counter As Integer

        ArabicConsole.WriteLine("** CalleeSub *********")
        For counter = 0 To ST.FrameCount - 1
            Dim SF As StackFrame = ST.GetFrame(counter)
            ArabicConsole.WriteLine(SF.GetMethod.Name & " ()")
        Next
    End Sub
End Module
```

مخرجات الشيفرة السابقة ستشمل الإجراء الرئيسي Main() وذلك لأنه اول إجراء يدخل طـــابور الاستدعاءات:

```
** CallerSub *********
CallerSub ()
main ()

** CalleeSub *********
CalleeSub ()
CallerSub ()
main ()
```

أسلوب أفضل:

بدلا من كتابة شيفرات الحلقة التكرارية في كل مرة تود معرفة طابور الاستدعاءات، لما لا تعرف إجراء خاص بك بالإسم GetStack() يعود بمصفوفة حرفية من النوع String تمثل الإجراءات في طابور الاستدعاءات:

```
Function GetStack() As String()
    Dim counter As Integer
    Dim stackArray() As String
    Dim ST As New StackTrace()

    For counter = 1 To ST.FrameCount - 1
        Dim SF As StackFrame = ST.GetFrame(counter)
        ReDim Preserve stackArray(counter)
        stackArray(counter - 1) = SF.GetMethod.Name
    Next

    Return stackArray
End Function
```

ملاحظة

لاحظ ان الحلقة في الإجراء ()GetStack السابق تبدأ بواحد، وذلك لاني اريد تجاهل العودة باسم الإجراء ()GetStack نفسه في المصفوفة.

الانعكاس Reflection هو أسلوب يمكنك من الاستفادة من أنواع البيانات لحظة التنفيذ وليس التصميم، بحيث تتمكن من إنشاء كائنات بمجرد الحصول على أسمائها نصيا. يعيب الانعكاس انه يتوجب عليك التحقق من كافة التفاصيل الدقيقة المتعلقة بالبيانات وأعضائها قبل استخدامها، ولكنه يمكنك من إنجاز مهام لا تخطر على بال احد، ولا تنسى ان البنية التحتية لإطار عمل .NET Framework تستخدم فئات الانعكاس (لعل أبرزها عمليات تسلسل الكائنات Object Serialization). لنودع الآن المشاريع من النوع Console Application، وننتقل إلى مرحلة جديدة –طالما انتظرناها- وهي **تطوير تطبيقات Windows** عنوان الجزء الثالث من هذا الكتاب.