

برمجة اطار عمل .NET باستخدام

Visual Basic .NET



-- تركي العسيري



شبكة المطورين العرب



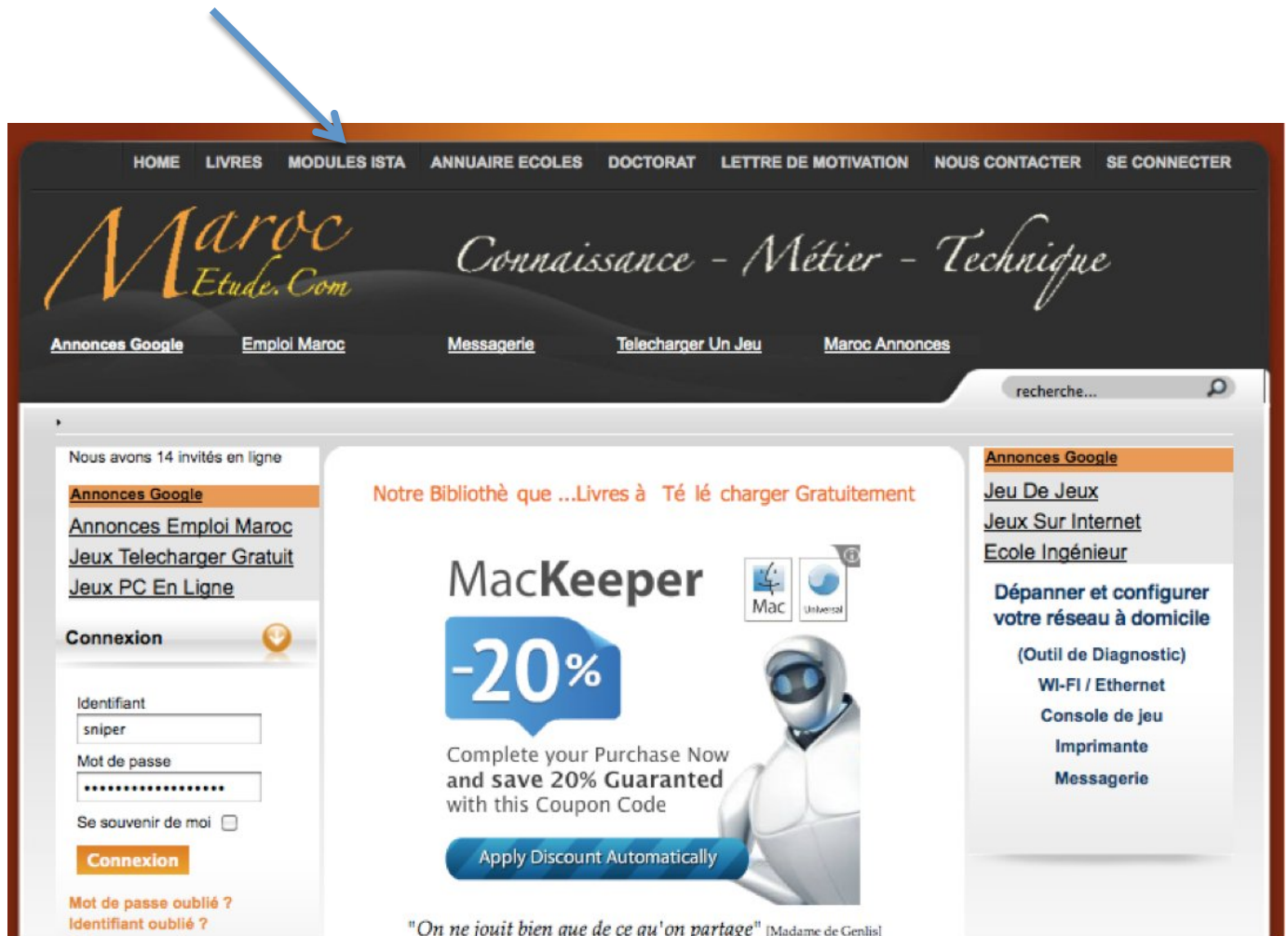
Arabian Developer Network

PORTAIL DE LA FORMATION PROFESSIONNELLE AU MAROC

Télécharger tous les modules de toutes les filières de l'OFPPT sur le site dédié à la formation professionnelle au Maroc : www.marocetude.com

Pour cela visiter notre site www.marocetude.com et choisissez la rubrique :

MODULES ISTA



بِسْمِ اللَّهِ الرَّحْمَنِ الرَّحِيمِ

((سُبْحَانَكَ لَا عِلْمَ لَنَا إِلَّا مَا
عَلَّمْتَنَا إِنَّكَ أَنْتَ الْعَلِيمُ الْحَكِيمُ))

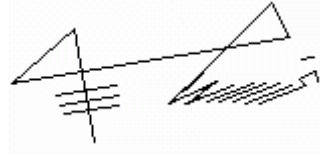
إهداء

الجمال في الحياة شيء يجبر الفؤاد
على ذكره في كل لحظة من لحظات خفقانه ...

وان لم تكوني من الجمال في الحياة،
فحسبي أن الجمال قد نبض إلى الحياة منك ...

أمي الحبيبة،
أهديك هذا الكتاب ...

ابنك المخلص



برمجة إطار عمل .NET باستخدام Visual Basic .NET

الطبعة الاولى 2003

- حقوق كتاب برمجة إطار عمل .NET. باستخدام Visual Basic .NET محفوظة للمؤلف، ولا يحق لأي شخص أو جهة رسمية من إعادة نشر هذا الكتاب أو جزء منه بأي وسيلة دون الإذن الخطي من المؤلف. وإلا فإنه سيكون معرضا للمطالبة بالتعويض وتطبيق أقصى العقوبة عليه.

- أسماء البرامج أو التقنيات أو الشركات (كـ Visual Basic .NET، ADO .NET، Microsoft... الخ) هي علامات تجارية مسجلة لأصحابها، والمؤلف يحترم هذه العلامات ويقر بها لمالكيها سواء كانوا أفراد أو شركات أو أي جهات تنظيمية أخرى، ولم يتم ذكرها للاختصار.

- أسماء الأشخاص أو الشركات والمذكورة في أمثلة هذا الكتاب هي أسماء وهمية ولا يقصد بها تحديد هوية أشخاص أو جهات معينة.

- تم اختبار المادة العلمية في هذا الكتاب والتحقق منها ومراجعتها، إلا أن المؤلف غير مسئول بأي شكل من الأشكال عن الأضرار الناتجة من تطبيق المعلومات في هذا الكتاب.

- جميع الآراء الموجودة في هذا الكتاب تعبر عن رأي المؤلف الشخصي حتى لو لم توثق بأمثلة أو أدلة حسية.

- فسح وزارة الاعلام رقم: 3876 -

المحتويات

تمهيد	١
تقديم	١
شكر وتقدير	ج
المقدمة	٥
لمن هذا الكتاب؟	و
ماذا عن مبرمجي Visual Basic 1 إلى 6؟	و
المصطلحات تعريب أم تعذيب؟	ز
ماذا يقدم لك هذا الكتاب؟	ح
القرص المدمج CD-ROM	ط
صفحة الكتاب على الانترنت	ي
الأخطاء (هام جدا)	ي
كلمة أخيرة	ي

الجزء الأول: الأساسيات	1
الفصل الأول: تعرف على Visual Basic .NET	3
الحياة قبل .NET	3
البرمجة تحت DOS	4
الانتقال الى Windows	4
الحلم أصبح حقيقة مع COM	7
تحديات الانترنت	8
عشرات التقنيات لأداء الوظائف	10
الحياة بعد .NET	10
الاستقلالية عن منصات العمل	11
.NET نسخة محسنة من COM	11
تكامل لغات البرمجة	13
خدمات ويب هي مستقبل الانترنت	13
ماذا عن المبرمج العربي؟	14

15	محتويات إطار العمل .NET Framework
16	الترجمة على الفور JIT
17	المجمعات Assemblies
17	بيئة التطوير .NET Visual Studio
18	نوافذ بيئة التطوير
24	القائمة الرئيسية
26	أشرطة الأدوات
26	كتابة برنامجك الأول
26	الحلول والمشاريع
28	أنواع المشاريع
30	بناء برنامجك الأول
31	استخدام ArabicConsole
32	الترجمة والتوزيع
35	الفصل الثاني: لغة البرمجة
35	الوحدات البرمجية Modules
38	الإجراء Sub Main()
39	الإجراء Sub New()
41	المتغيرات والثوابت
41	التصريح عن المتغيرات
43	قابلية الرؤية وعمر المتغيرات
48	أنواع البيانات
52	إسناد القيم
58	الثوابت
59	التركيبات والمصفوفات
59	التركيبات من نوع Enum
62	التركيبات من نوع Structure
67	المصفوفات
69	الإجراءات والدوال
70	الإرسال بالمرجع أو القيمة

72	تخصيص الوسيطات المرسلة
73	تجاوز الحدود مع Windows API
74	التفرع والتكرار
74	التفرع باستخدام If ... Then
77	التفرع باستخدام Select Case
79	الحلقات التكرارية
81	مجالات الاسماء
82	تعريف مجال اسماء
84	الوصول الى عناصر مجال الاسماء
85	استيراد مجال اسماء باستخدام Imports
87	استيراد مجال اسماء دون استخدام Imports
89	الفصل الثالث: الفئات والكائنات
89	مدخلك السريع للفئات
94	بناء اعضاء الفئات
94	الحقول Fields
96	الطرق Methods
105	الخصائص Properties
110	الأحداث Event
116	استخدام الكائنات
116	ما هي حقيقة الكائن؟
119	عبارات خاصة بالكائنات
123	اسناد القيم
126	حياة وموت الكائنات
137	ارسال الكائن بالمرجع او القيمة
138	الأعضاء المشتركة
138	الحقول المشتركة Shared Fields
140	الطرق المشتركة Shared Methods
141	الخصائص المشتركة Shared Properties
142	الاحداث المشتركة Shared Events

147	الفصل الرابع: الوراثة
147	مقدمة الى الوراثة
147	مبدأ الوراثة
149	تطبيق الوراثة بـ Visual Basic .NET
152	التعامل مع الفئات الوارثة والمورثة
152	وراثة الأعضاء
154	المشيدات Constructors
157	التعامل مع الكائنات
159	إعادة القيادة Overriding
161	اعادة قيادة الطرق والخصائص
166	استخدام MyBase
168	استخدام MyClass
169	التظليل Shadowing
173	الاعضاء المشتركة Shared Members
174	كلمات محجوزة إضافية
174	الكلمة المحجوزة NotInheritable
174	الكلمة المحجوزة MustInherit
176	الكلمة المحجوزة MustOverride
177	محددات الوصول
177	قابلية الرؤية للفئات
181	قابلية الرؤية لاعضاء الفئات
184	تأثير محددات الوصول على المشيدات
187	الفصل الخامس: الواجهات، التفويض، والمواصفات
187	الواجهات
190	بناء واجهة
192	تضمين الواجهة
196	الوصول الى الواجهة
197	وراثة الواجهات
198	واجهات من اطار عمل .NET Framework

199	IComparable	الواجهة
201	ICloneable	الواجهة
203	IEnumerator و IEnumerable	الواجهتان
207		التفويض
208		الاجراءات الستاتيكية
211		اجراءات الفئات
212		محاكاة الاحداث
214		دمج التفويضات
217		المواصفات
217	Visual Basic .NET	صيغة كتابة المواصفات في
218NET Framework	مواصفات من اطار عمل
218	Conditional Attribute	المواصفة
221	DebuggerStepThrough Attribute	المواصفة
221	Obsolete Attribute	المواصفة
223	FieldOffset و StructLayout	المواصفة
225		بناء مواصفات خاصة

229 الجزء الثاني: إطار عمل .NET Framework

231 الفصل السادس: الفئات الأساسية

231	System.Object	الفئة
232	Object	طرق الفئة
234		البيانات المرجعية والبيانات ذات القيمة مرة اخرى
236		الصندوق واللاصندوق
236		الفئات الحرفية
239		الخصائص والطرق
242		مقارنة الحروف
243	CultureInfo	الفئة
246		البحث عن الحروف
247	Char	الفئات من النوع

248	الفئات من النوع StringBuilder
250	الفئات العددية
251	الخصائص والطرق
252	تنسيق الاعداد
254	الفئة Math
255	توليد الاعداد العشوائية Random Numbers
256	فئات اخرى
256	فئات الوقت والتاريخ
263	الفئات من النوع Enum
265	الفئات من النوع Array
270	مجال اسماء System.Collections
270	الواجهات ICollection و IList
271	الفئة Stack
272	الفئة Queue
273	الفئة ArrayList
275	الفصل السابع: اكتشاف الأخطاء
275	فكرة عامة
275	اخطاء وقت التصميم
276	اخطاء وقت التنفيذ
278	الشوائب
279	الكائن Exception
280	تفادي الاستثناءات Catching Exceptions
284	رمي الاستثناءات Throwing Exceptions
287	انشاء فئات استثناءات خاصة Custom Exceptions
288	الكائن Err
288	تفادي الاستثناءات
290	رمي الاستثناءات
290	الاختيار بين Exception و Err
292	ادوات التنقيح من Visual Studio .NET

292	اساليب التنفيذ
294	نوافذ اخرى
296	Configurations الاعدادات
299	الفصل الثامن: الملفات والمجلدات
299	Directory الفئة
301	طرق تعود بمسارات
302	البحث عن الملفات والمجلدات
303	File الفئة
305	Stream الفئة
306	الخصائص والطرق المشتركة
309	التعامل مع الملفات النصية
312	التعامل مع الملفات الثنائية
314	تكوين Custom Streams خاصة
316	فئات اخرى
316	Path الفئة
317	DirectoryInfo و FileInfo الفئات
321	الفصل التاسع: تسلسل الكائنات Object Serialization
322	مدخلك الى تسلسل الكائنات
322	ما هو التسلسل؟
323	التسلسل بالصيغة الثنائية Binary Serialization
325	تسلسل انواع بيانات مخصصة (غير قياسية)
228	Object Graph خريطة الكائنات
231	نسخ الكائنات
335	انشاء Custom Serialization خاصة
336	ISerializable الواجهة
337	مثال تطبيقي
338	التسلسل بصيغة XML
340	XmlSerializer الفئة
341	مواصفات اضافية

345 احداث تقع عند عكس التسلسل
349 الفصل العاشر: مسارات التنفيذ Threading
349 مقدمة الى مسارات التنفيذ
350 انواع مسارات التنفيذ
351 متى تستخدم مسارات التنفيذ المتعددة؟
353 انشاء مسار تنفيذ
354 الطرق والخصائص
364 التعامل مع مسارات التنفيذ
368 مشاركة البيانات
369 المتغيرات المحلية الديناميكية
370 ThreadStatic Attribute الموصاف
372 TLS وحدة التخزين المحلية
374 تبادل البيانات بين مسارات التنفيذ
377 Thread Synchronization التزامن
377 SyncLock ... End SyncLock التركيب
379 Synchronization الموصاف
380 MethodImpl الموصاف
381 فئات اخرى
383 ThreadPool الفئة
386 Timers المؤقتات
387 System.Timers.Timer المؤقت
388 System.Threading.Timer المؤقت
391 الفصل الحادي عشر: المجمعات Assemblies
391 Managed Modules الوحدات المدارة
392 Assemblies المجمعات
393 المجمعات الاحادية والمتعددة الملفات
394 اساليب تنفيذ المجمعات
395 المجمعات الخاصة والمشاركة
397 Strong Names الاسماء القوية

397	المواصفة Assembly
398	ملفات التهيئة Configuration Files
399	انواع ملفات التهيئة
400	تغيير الاعدادات
400	اعدادات لملفات التهيئة
403	استخدام الاداة .NET Framework Configuration
405	ادوات الترجمة، الربط، والتسجيل
406	المترجم VBC.EXE
408	الرابط AL.EXE
412	المسجل SN.EXE
413	المسجل GACUTIL.EXE
415	الفصل الثاني عشر: فئات الانعكاس Reflection Classes
415	التعامل مع المجمعات والوحدات المدارة
416	الفئة Assembly
418	الفئة Module
419	التعامل مع انواع البيانات
419	الفئة System.Type
421	خصائص اضافية
422	التعامل مع الاعضاء
422	الفئة القاعدية MemberInfo
425	التعامل مع الحقول
426	التعامل مع الخصائص
428	التعامل مع الطرق
429	التعامل مع الاحداث
430	الوسيطات Parameters
431	التعامل مع الكائنات
431	الفئة ReflectionExample
432	اسناد/قراءة قيم الحقول
433	اسناد/قراءة قيم الخصائص

433	استدعاء الطرق
435	مواضيع اخرى
435	الانشاء الديناميكي للكائنات
436	معرفة الاجراءات المستدعية
439	الجزء الثالث: تطوير تطبيقات Windows
441	الفصل الثالث عشر: نماذج Windows Forms
442	مدخلك الى نماذج Windows Forms
442	مصمم النماذج Form Designer
445	نظرة حول الشيفرة المولدة
448	التعامل مع اكثر من نموذج
449	محل الفئة Form من الاعراب
450	الخصائص، الطرق، والاحداث
450	خصائص النموذج
453	طرق النموذج
456	احداث النموذج
460	نماذج MDI Forms
461	النوافذ الابناء Child Windows
462	خصائص وطرق اضافية
463	القوائم Menus
465	الخصائص، الطرق، والاحداث
466	القوائم المنبثقة Popup-Menu
466	نماذج MDI مرة اخرى
469	الانشاء الديناميكي للقوائم
469	مواضيع متقدمة
469	التفاعل مع نوافذ Modeless
470	وراثة النماذج Form Inheritance
474	النماذج المحلية
479	الفصل الرابع عشر: الأدوات Controls

479 الخصائص المشتركة
480 اسم الاداة Name
480 خصائص المظهر
482 خصائص الموقع والحجم
487 خصائص الاحتضان
488 خصائص الالوان
489 خصائص التركيز
489 خصائص الجدولة
490 خصائص اخرى
490 الطرق المشتركة
491 الاحداث المشتركة
492 احداث الفأرة
493 احداث لوحة المفاتيح
494 احداث التركيز
494 احداث اخرى
495 عرض سريع للادوات
497 الاداة Label
497 الاداة LinkLabel
498 الاداة TextBox
501 الاداة Button
501 الاداة CheckBox
502 الاداة RadioButton
502 الاداة ListBox
504 الاداة CheckedListBox
505 الاداة ComboBox
505 الاداة ImageList
506 الاداة TreeView
508 الاداة ListView
509 الاداتان StatusBar و ToolBar

510	الاداة Splitter
510	ادوات صناديق الحوار الشائعة
514	ادوات المزودات
515	ادوات اخرى
518	تقنية المرآة
518	الخاصية RightToLeft
520	قصور الخاصية RightToLeft
521	مدخلك الى تقنية المرآة
522	تطبيق تقنية المرآة بـ Visual Basic .NET
527	مشاكل اضافية
527	ادوات صناديق الحوار الشائعة
528	صناديق الرسائل
531	الفصل الخامس عشر: مبادئ GDI+
531	الرسم المتقدم
532	الكائن Graphics
533	رسم الخطوط، المستطيلات، والدوائر
534	رسم المنحنيات المعقدة
535	كائن القلم Pen
538	كائن مسار الرسم GraphicsPath
537	التعبئة
540	كائن الفرشاة Brush
542	انظمة القياس
545	التعامل مع الصور
545	تحميل وحفظ الصور
546	عرض الصور
549	عكس، قلب، وسحب الصور
552	تحديد الالوان
552	الرموز Icons
554	المخرجات النصية

554	عوائل الخطوط
556	رسم النصوص
557	التفاف النص
558	الكائن StringFormat
567	عودة الى الادوات Controls
571	الفصل السادس عشر: مواضيع متقدمة
571	تطوير ادوات خاصة
572	وراثة اداة
578	حضان مجموعة من الادوات
581	انشاء اداة مستقلة
582	لمسات فنية اضافية
587	تطوير خدمات Windows
587	مقدمة الى خدمات Windows
588	انشاء مشاريع من نوع Windows Service
590	تصحيح الشيفرة
592	الفئة System.IO.FileSystemWatcher
593	كتابة الشيفرات
595	تسجيل الخدمة
596	الاداة InstallUtil.EXE
598	فئات اخرى
598	الفئة Application
599	الفئة Cursor
600	الفئة SendKeys
601	الفتتان RegistryKey و RegistryKey
603	الفئة Help
605	الجزء الرابع: برمجة قواعد البيانات
607	الفصل السابع عشر: استخدام ADO.NET
608	مدخل الى ADO.NET

608	الوضع المتصل والوضع المنفصل
609	مزودات .NET Data Providers
611	فئات ADO.NET
612	كائن الاتصال Connection
612	انشاء كائن اتصال
613	نص الاتصال
614	فتح واغلاق الاتصالات
617	كائن الاوامر Command
618	انشاء كائن اوامر
618	الربط مع اتصال
620	تنفيذ جمل الاستعلام SQL
621	قراءة السجلات
623	كائن البيانات DataReader
623	انشاء كائن بيانات
624	قراءة السجلات
626	خاص بمستخدمي Microsoft SQL Server ®
627	قراءة نتائج متعددة
629	الفصل الثامن عشر: ADO.NET للوضع المنفصل
629	كائن البيانات DataSet
632	الفئة DataTable
633	الفئة DataRow
634	الفئة DataColumn
634	الفئة DataRelation
635	انشاء كائن بيانات DataSet من الصفر
638	كائن المحول DataAdapter
638	سيناريو الوضع المنفصل
639	انشاء كائن محول DataAdapter
640	الربط مع اتصال
641	قراءة البيانات

644	تحديث البيانات
648	تخصيص افضل للخصائص xxxCommand
649	اتقاء شر التعارضات
650	عرض التعارضات
651	الحدث RowUpdated
653	الفصل التاسع عشر: ربط البيانات والتكامل مع XML
653	ربط البيانات
654	انواع الربط
655	ميكانيكية الربط
656	الربط الى مصفوفة
659	الربط باستخدام ADO.NET
660	الربط المعقد Complex Binding
662	فئات خاصة بـ XML
663	الفئة XmlTextReader
664	الفئة XmlTextWriter
665	تكامل ADO.NET و XML
666	كتابة البيانات بصيغة XML
668	قراءة البيانات بصيغة XML

669 **الجزء الخامس: برمجة ويب**

671 **الفصل العشرون: تطبيقات ASP.NET (1)**

671	الخادم IIS
672	تركيب الخادم IIS
673	الادلة الوهمية
675	الوصول الى الادلة الوهمية
677	مدخلك الى نماذج Web Forms
678	انشاء المشروع
681	ضبط الاعدادات الرئيسة
682	كتابة الشيفرات

685	تحليل الشيفرة
688	اساليب تنفيذ الصفحة
690	الخلاصة
690	الفئة Page
690	خصائص صفحة النموذج
693	طرق صفحة النموذج
693	احداث صفحة النموذج
694	وسوم اضافية
695	الادوات
695	ادوات Web Forms Controls
696	ادوات HTML Forms Controls
697	ادوات التحقق Validation
701	الفصل الحادي والعشرون: تطبيقات ASP.NET (2)
701	كائنات صفحات ASP.NET الاساسية
701	الكائن HttpRequest
703	الكائن HttpResponse
705	الكائن HttpServerUtility
705	الكائن HttpSessionState
706	الكائن HttpSessionState
707	الملف Global.asax
707	الاجراءات xxxStart() و xxxEnd()
708	الاجراء Global_Error()
709	الامان Security
709	مدخلك الى الصلاحيات
708	اوضاع التصديق في ASP.NET
709	ملفات التهيئة
709	الوضع Forms للتصديق
713	الوسم <credentials>
714	صفحة تسجيل الدخول Login

715 مواضيع متقدمة
715 التخزين Caching
718 المتغيرات العامة
718 حماية الصور
720 التوليد الديناميكي للصور
723 Web Services ويب خدمات والعشرون: الفصل الثاني
723 مدخلك الى خدمات ويب
724 كيف تعمل خدمات ويب؟
725 بناء خدمة ويب
725 انشاء المشروع
727 كتابة الشيفرة
728 اختبار الخدمة من المتصفح
731 استخدام الخدمة
735 تحديث الخدمة

الملاحق

1 م الملحق أ: لغة وصف البيانات XML
7 م الملحق ب: لغة الاستعلام SQL

تقديم

لقد أسعدني خبر أن تركي العسيري قام بكتابة كتابا جديدا عن لغة .NET Visual Basic والتي تعتبر لغة المستقبل للمطورين والمبرمجين، وان كنت لا تعلم من هو تركي العسيري، فدع هذا الكتاب يعرفك به؛ يكفيك مقالاته التي تمتلئ بها المواقع العربية حول تقنية .NET، ولا تنسى انه المؤسس للموقع الشهير **شبكة المطورون العرب** (dev4arabs.com) وكاتب لجميع الشيفرات المصدرية التي تنبض الموقع بالحياة، وما يجعلني أشعر بالفخر هو أن الكاتب عربي ويوجه خبرته ومعرفته للعرب.

يتمتع الأخ تركي بمزايا خاصة سواء كان مبرمجا أو كاتبا، فلهذه من الأسلوب الذي يجعلك تستوعب الأمور المعقدة وكأنها معادلة $2=1+1$! والآن أرى أنه أنجز مشروعا (كما يلقب كتابه) يعتبر أول كتاب من مؤلف عربي حول تقنية .NET Microsoft، كما سطر مجموعة من الصفحات والموجه خصيصا للمطور العربي.

كخبير تصميم وتطوير مواقع، أرى بأن الكتاب سيفيد الكثير من المبرمجين العرب التائهين في تقنية .NET، فهذه التقنية تعتبر تقنية المستقبل لتطوير التطبيقات ومواقع الإنترنت، وهي قوية جدا ولها من الإمكانيات ما يفوق التوقعات، ولكن مسألة استيعابها سيكون صعبا على اغلب المبرمجين المستجدين والمخضرمين أيضا، هذا وان علمت أنها من أحدث التقنيات الصادرة من معامل ريدمون بولاية واشنطن (اعني من شركة Microsoft)، ووجود مصدر لإتقانها يحتاج إلى جهد جهيد، ومن النادر الحصول على مرجع لها باللغة العربية، فكل الكتب العربية والخاصة بتقنية .NET مترجمة، فما بالك إذا كان الكاتب عربي يعرف كيف نفكر وما هي متطلباتنا وإلى ماذا نتطلع أو نتوقع من كتاب بهذا الحجم والتخصص.

يتميز هذا الكتاب بالنمط الاستيعابي الذي يتعمده المؤلف، فهو يركز على السهولة والمنطق في التسلسل لإيصال المعلومة للقارئ وهذا ما يعرف عن الكاتب، فأسلوبه سلس ودائما ما يحاول النزول لمستوى القارئ ثم يرتقي به كلما استمر في القراءة، كما انه يضيف على الكتاب أسلوب المحاوره مما يوحي أن المؤلف أمامك ويناقشك في المواضيع وكأنك في قاعة محاضرة، أضف

إلى ذلك الأمثلة الفورية التي يعرضها عند الحديث بعد شرح كل نقطة، فهي ليست كبيرة ليصعب تتبعها.

أما عن المسألة التجارية، فواضح من الأسلوب في الكتابة ووفرة المعلومات بأن الكاتب حريص على إيصال أكثر قدر من المعرفة لجعل القارئ يتعلم أكبر قدر ممكن وبالتالي يستطيع تطبيق ما تعلمه والاستفادة من قراءة الكتاب، فلا يقوم بنسخ جداول ووثائق NET. Documentation للتكثير من عدد الصفحات وزيادة الدخل كما تفعل اغلب الكتب المنتشرة في الأسواق، فمعظم هذه الكتب تتجاوز صفحاتها الألف صفحة، وبعد شرح كل نقطة يقوم المؤلف بعرض جدول منسوخ من وثائق NET Documentation. وإضافته. ولكن مع هذا الكتاب فلن يحدث معك ذلك، حيث أنك ستلاحظ جملة المؤلف المتكررة "راجع مكتبة MSDN لمزيد من التفاصيل" وهذا خير دليل على أن الكاتب لا يريد زيادة الصفحات المنسوخة من وثائق NET. Documentation وتكرارها في كتابه.

وختاماً أتمنى من الله أن يوفق المؤلف في نشر المعرفة للعرب، وأن يبدأ الشباب بالمضي قدماً في مجال النشر والتعليم لأنه سلاح العصر، وأنا واثق كامل الثقة بأنك لن تشعر بالندم بعد اقتنائك لهذا الكتاب، ولن تضعه مع الكتب الأخرى في رفوف مكتبك، بل سيضل ناصباً على سطح مكتبك.

تنبيه أخير ناتج من تجربة شخصية: الكثير من المفاجئات والأفكار ستجدها في هذا الكتاب، لذلك لا تقرأ هذا الكتاب قبل موعد النوم بدقائق حيث أنك لن تستطيع النوم دون تجربة الشيفرات المصدرية بنفسك!

إياد يحيى مكي زكري،

عضو في الإتحاد العالمي لمصممي ومدراء المواقع (IAWMD) (iawmd.com).

شكر وتقدير

مجموعة من الأشخاص دعموني بشكل مباشر و غير مباشر لإنجاز كتاب **برمجة إطار عمل .NET باستخدام Visual Basic**. صحيح أن كلمة شكر قد لا تكون كافية للأشخاص التالي ذكر أسمائهم، ولكن ما كان في القلب أعظم.

لنبدأ بالصديق إباد مكي كاتب صفحة **التقديم** في هذا الكتاب، أشكرك على كلماتك الجميلة - والتي قد بالغت في بعضها حولي، كما اقدر عرفانك ووقفات قريبا مني في الفترات السابقة، وبالنسبة لنصائحك وتوجيهاتك فقد أخذتها بعين اعتباري من الصفحة الأولى حتى الأخيرة عند كتابة هذا الكتاب.

اختص بالشكر الجزيل جدا جدا للأستاذ الفاضل سالم المالكي، والذي بنى معي أساسيات هذا الكتاب، كما غطى على أبرز الأخطاء التي اتبعتها منذ البداية والذي علمني كيف تكون البداية الصحيحة لتعليم لغات البرمجة -خاصة إن كانت جديدة كليا.

ثلاثة أسباب تجعلني اشكر الصديق محمد الحلبي، السبب الأول هو مراجعته الدقيقة لكل حرف من حروف شيفراتي المصدريّة (رغم انه مبرمج .NET C# Visual وليس Visual Basic)، والثاني وقفاته قريبا مني رغم انشغالاته وظروف السفر التي لم تمنعه من مواصلة المشوار معي، أما السبب الثالث فهو تكلفه بتوفير Visual Studio .NET وإرسالها لي، ولو لم يفعل ذلك فلم يكون هذا الكتاب أمام عينيك.

اعتقد انه من المناسب هنا ذكر مشرفي **شبكة المطورون العرب** (dev4arabs.com) (الموقع الذي أديره مع مجموعة من المحترفين العرب في مختلف مجالات علوم الحاسب) وذلك نظير استمرارهم في إدارة الموقع بينما كنت مشغولا في كتابة هذا الكتاب، اشكر كلا من: طارق موسى، محمد نسمان، محمد العتيبي، أيمن المدهون، عمر رضوان، لمياء حاشي، هند محسن، محمد لبد، واحمد الشمري.

كما يتحتم علي هنا شكر المهندس حسين فاضلي الذي سهل لي عقبة توفير المراجع اللازمة حول Visual Basic .NET وتقنية Microsoft .NET (في أيامي الأولى مع هذا الكتاب لم تكن متوفرة بكثرة في الأسواق). كما اشكر الأستاذ عبدالله العتيق مؤسس موقع (vb4arab.com) (أكبر موقع عربي للغة Visual Basic) على دعمه المعنوي، ليس فقط في الإعلان عن كتابي بل

تخصيص صفحة خاصة به. ولا أنسى شكر الأستاذ دانيال ريد (حيث انه أكثر سبقا مني في مجال تأليف كتب البرمجة) على إرشاداته الفائقة الروعة لطريقة إعداد مخطط العمل لكتابة هذا الكتاب. من الأسماء الذي ساهمت في هذا الكتاب أيضا، الصديق صالح الغامدي مصمم غلاف الكتاب، وهناك أيضا فهد العمير الذي سهل لي عملية عرض أمثلة هذا الكتاب واختصارها حتى يتم استيعابها بسهولة. وأما البعيدة القريبة الأستاذة الفاضلة عادة فلا اعلم من أين ابدأ أو انتهى لعرض فضائلها وجمالها علي والتي تحتاج إلى كتاب كامل لذكرها. إلى جميع الأسماء السابق ذكرها، اكرر شكري الجزيل وامتناني الكبير لكم، المشاركة والعمل معكم في الفترة الماضية كان فرصة من اسعد الفرص في حياتي، كما اعتقد أنني شخص محظوظ جدا بمعرفتكم الطيبة، عسى أن تجمعنا أعمال أخرى مشرفة. وأخيرا، مهما سطرت من كلمات الشكر فإنها لم ولن توفي حق أعظم من رأته عيني في حياتي، إليك يا صانع الرجال، إليك يا سيدي، إليك يا والدي العظيم، كم سجدت لله شكرا وحمدا على نعمته علي في أن شرفني وجعلني من صلبك.

-- تركي

المقدمة

عندما شرعت في كتابة هذا الكتاب، خائنتني العبرة ولم تأتي التوقعات كما رسمت في الخاطر، فالموضوع اكبر من كونه شرح لغة برمجة وإجراء تطبيقات بها، حيث ان كتابة كتاب عن لغة البرمجة .NET Visual Basic ليس سوى لهجة بلسان آخر تخاطب بها تقنية Microsoft .NET.

سواء كنت مبرمج .NET Visual Basic او مبرمج .NET Visual C# او اي لغة برمجة أخرى موجه الى إطار العمل .NET Framework. فلن تقدم ولن تؤخر ذلك الشيء الكبير، إذ ان اللقب الذي عليك البحث عنه دائما هو **مبرمج .NET**. او -لمزيد من التفصيل- مبرمج .NET. بلهجة .NET Visual Basic. وان لم يتضح لك المعنى من اللقب مبرمج .NET. فعليك معرفة ان لغة البرمجة .NET Visual Basic ليست سوى مفتاح بسيط تستخدمه للباب المعقد والمسمى .NET Framework، فالاحتراف في برمجة .NET Visual Basic لن يتطلب منك ذلك الجهد للوصول إليه في أيام معدودات (قراءة الفصول الخمس الأولى من هذا الكتاب كافية إلى حد كبير)، أما الاحتراف في تقنية .NET. فيحتاج إلى تفرغ ذهني من كل شيء في حياتك تقريبا.

إطار عمل .NET Framework. معقد جدا، وهو بحر ليس له بداية ولا نهاية، وقد لا أبلغ ان قلت ان كل ميناء من مواني هذا البحر تحتاج إلى كتاب مخصص. فألاف الصفحات والخاصة بمستندات تقنية .NET. Microsoft الرسمية خير دليل. ولتقديم كتاب عن هذه التقنية، كان علي اختيار احد الحلول الثلاث:

الأول هو كتابة كتب متعددة يختص كل واحد فيها بمجال معين، كتاب عن تطوير تطبيقات Windows وآخر عن برمجة ويب، وثالث عن الاستخدام المتقدم لتقنية ADO.NET لبرمجة قواعد البيانات، ولكن يعيبه الوقت الطويل الذي سيستغرقه بالإضافة ان جميعها تتطلب شرح أساسيات اللغة، مما يعني تكرار معظم الصفحات. الحل الثاني، هو مشاركة كتاب آخرين ومحاولة توسعة وزيادة المادة العلمية في هذا الكتاب، ولكن يعيبه اختلاف الأساليب التي يتبعها كل مؤلف مما قد يسبب التشويش على القارئ. أما الحل الثالث فهو اقتطاف من كل بستان زهرة ووضعها كمقدمة ومدخل لك يمكنك من الانطلاق منه، وهذا ما فعلته في كتاب **برمجة إطار عمل .NET**.

باستخدام .NET Visual Basic.

لمن هذا الكتاب؟

هذا الكتاب للمبتدئين ام للمتوسطين او للمحترفين؟ سؤال يراود الكثير ويهتمون في إجابته أكثر من اهتمامهم بمؤلف الكتاب. لكل إنسان مقياس خاص لتصنيف المستويات، وقد يكون مقياسي الشخصي مرفوض من قبل الكثيرين، لذلك لن أعطيك إجابة مباشرة لهذا السؤال، وسأكتفي بذكر صفات الأشخاص الذين اعتقد ان الكتاب سيكون مناسب لهم من نظرتي الشخصية.

أستطيع تصنيف كتب لغات البرمجة إلى أربعة أنواع، النوع الأول وهو **التعليمي Tutorial** حيث يوجه إلى الأشخاص الذين ليس لديهم الوقت الكافي للقراءة، ولا يودون استيعاب المبادئ بقدر ما يهمهم إنجازها. فستجد في مثل هذه النوعية من الكتب عشرات السطور المتمثلة في خطوات Step by Step مرقمة تقوم بإنجاز مهمة معينة دون ذلك الشرح.

النوع الثاني هو **المراجع Reference**، وهي كتب لا تقرأ من الغلاف إلى الغلاف وإنما يرجع لها من وقت لآخر. وكما تعلم ان المراجع تكتب من قبل آلاف الأشخاص، وقد احتاج للكتابة بأصابع قدمي حتى أنجز مرجع قبل ان تشيب شعرات رأسي.

النوع الثالث من أنواع الكتب هو **ورشة العمل Workshop**، وهو عندما يميل الكتاب إلى الجانب التطبيقي أكثر من الشرح، ولكن يفترض المؤلف انك تعلم كل شيء عن لغة البرمجة، حيث يعرض لك الأساليب البرمجية المتعددة لإنجاز المهام وتطبيقها مباشرة.

أما الكتاب الذي أمام عينيك فهو من النوع الرابع وهو **الاستيعابي Comprehensive** فأميل فيه إلى شرح الأساسيات وبناء قواعد معرفية تمكنك من الانطلاق في برمجة إطار عمل NET Framework. من أوسع أبوابها، حيث يمكنك اعتبار ان كل فصل من فصول هذا الكتاب مقدمة او مدخل الى استخدام تقنية من تقنيات إطار عمل NET Framework، ويبقى الأمر في النهاية عليك لتتعمق وتتخصص في المجال الذي تريده بنفسك.

ماذا عن مبرمجي 6 و 1 Visual Basic؟

حسنا، دعني هنا أخطب تلك الفئة من المبرمجين عشاق النغمة الرنانة Visual Basic. العلاقة بين Visual Basic و الإصدارات السابقة 6 > Visual Basic 1 لا تكاد ان تكون إلا علاقة تشابه أسماء فقط، فهي مشروع تسويقي أكثر من ما هو تطوير للغة البرمجة المحكرة من

قبل Microsoft .NET. حيث ان الضريبة التي كانت ستكلف مشروع Microsoft .NET هو إنتاج لغة البرمجة الجديدة Visual C#.NET والاستغناء عن الملايين من مبرمجي Visual Basic حول العالم. إنتاج لغة البرمجة الجديدة أمر لا بأس منه، أما قضية الاستغناء عن ملايين المبرمجين فهي بحاجة بكل تأكيد - إلى إعادة نظر .

بوضوح وصراحة مباشرة، أنتجت Microsoft لغة البرمجة الجديدة Visual C#.NET. Visual C# وأكد أن خيل احد صناع القرار في تلك الشركة يقول: لما لا نقوم باستبدال صيغ لغة C# إلى صيغ شبيهة بلغة BASIC، أي باختصار - احذفوا الأقواس من عبارة الشرط if واجبروا كتابة Then بعدها، مع إلغاء ضرورة استخدام الفاصلة المنقوطة ";" نهاية كل سطر .

صحيح ان Visual Basic .NET هي نسخة بلسان آخر من Visual C#.NET، إلا أن الفروق طفيفة ولا تكاد تذكر (أستطيع ان اجزم ان Microsoft تعمدت وضع هذه الفروق حتى نقتنع أنهما لغتا برمجة مختلفتين).

بالرغم من تشابه الصيغ بين Visual Basic .NET والإصدارات السابقة Visual Basic 6->1 إلا انه من الخطأ الكبير والجرم العظيم اعتبار Visual Basic .NET إصدار جديد منها، حيث انك ستتعامل مع لغة برمجة جديدة كلياً وليس لها اي علاقة -كما ذكرت- مع الإصدارات السابقة من Visual Basic .NET. لذلك، تقبل نصيحتي هذه قبل ان تعاني الكثير من المتاعب -كما عانيت انا- وانسى كل ما تعلمته سابقاً في الإصدارات القديمة من Visual Basic، وضع في عين اعتبارك دائماً انك تتعامل مع لغة برمجة جديدة وحديثة العهد اسمها Microsoft Visual Basic .NET.

المصطلحات تعريب ام تعذيب؟

لا تزال مشكلة المصطلحات العربية في رمتها، وبما أنني لست في منصب مسئول لتحديد واختيار الترجمات العربية الصحيحة للمصطلحات الأجنبية، فذلك يعني حريتي في اختيار ما أراه مناسباً ورفض ما لا يناسبني.

وجهة نظري الشخصية حول ترجمة المصطلحات تتمحور حول اختيار الكلمة الأكثر شعبية أو التي تميل إلى توضيح المعنى التقني وليس المعنى الحرفي، لديك الكلمة **Help** مثلاً والتي نترجم بشكل صحيح - إلى "مساعدة"، ولكني فضلت اختيار المصطلح "تعليمات" لشعبيته بين

مستخدمي نظم Windows. من ناحية أخرى، لديك المصطلح **Overloading** والذي تترجمه الكتب العربية "بالتحميل الزائد"، وهو تعبير لم استطع تقبله لا من بعيد ولا من قريب، فلا يوحي معناه بالهدف منه، وكان اختياري للتعبير "إعادة التعريف" هو الأنسب والأفضل للترجمة التقنية له. مع ذلك، قدمت الأمر في البداية والنهاية لك أيها القارئ العزيز لاختيار المصطلحات التي تناسب توجهاتك اللغوية، فلا أكاد اذكر مصطلح إلا وأرفق المقابل الإنجليزي له من مستندات NET Documentation، وإن كان اختياري لا تناسب توجهاتك اللغوية، فيمكن الشطب على الكلمة وكتابة ما تريد بدالها.

ماذا يقدم لك هذا الكتاب؟

دعنا نضع النقاط على الحروف، ونكون واقعين قدر الامكان، فقبل ان توجه لي النقد حول قصوري في ذكر محتويات إطار عمل NET Framework، عليك معرفة ان هذا الكتاب ليس مرجعا من مراجع MSDN، فلا تتوقع مني شرح كل شيء وذكر كافة التفاصيل عن المواضيع التي تطرقت لها، فانك تخاطب شخص يكتب بعشرة أصابع فقط، ومن غير منطقي مقارنة مجهوده بمجهود آلاف الموظفين في شركة Microsoft. وإن كنت شخص تقدر الكم العددي على الكيف المعلوماتي، فيؤسفني إخبارك بان هذا الكتاب ليس مناسب لك.

إذا هل هذا الكتاب بهذا السوء؟ في الحقيقة ستكون شهادتي بكل تأكيد - مجروحة إن مدحته، ولكن دعني اعرض لك ماذا ستجد بين ثنايا الـ 700 صفحة والمكونة لهذا الكتاب:

أساسيات لغة البرمجة NET. Visual Basic من الصفر تبدأ بعرض كيفية كتابة أول برنامج لك، مع شرح الصيغ والعبارات المستخدمة في لغة البرمجة كجمل الشرط، التفرع، والتكرار. بالإضافة إلى نظرة كاتنية التوجه OOP وطرق تعريف الفئات Classes لإنشاء الكائنات Objects، والاستخدام الأمثل لها لتطبيق مبادئ الوراثة Inheritance وتعدد الواجهات Polymorphism.


عرض سريع لمكتبة فئات إطار عمل NET Framework. إن أردت اخذ جولة سريعة حول مكتبة فئات إطار عمل NET Framework، فستجد ملخصا لها هنا، حيث اعرض لك مجموعة من الفئات الأساسية، وفئات أخرى تستخدمها لإنجاز مهام معينة، مثل: كائنات الاستثناءات Exceptions، دخل خرج الملفات File IO، تسلسل الكائنات Object

Serialization، مسارات التنفيذ Threading، طريقة التعامل مع المجمعات Assemblies باستخدام فئات الانعكاس Reflection Classes.

أساسيات تطوير تطبيقات Windows Application باستيعاب فكرة نماذج Windows Forms وطريقة وراثتها، كما آخذك في جولة سريعة حول معظم الأدوات Controls والغرض منها، بالإضافة إلى تطبيق تقنية المرآة Mirroring عليها. ثم اخصص فصل كامل يعتبر مدخلك إلى استخدام تقنية GDI+ وطرق الرسم والتعامل مع الصور والمخرجات النصية. المزيد أيضا، اعرض لك كيفية تكوين أدوات خاصة Custom Controls وبناء خدمات Windows Services.

استخدام ADO.NET لبرمجة قواعد البيانات وشرح لفكرة الوضع المتصل Connected Mode والوضع المنفصل Disconnected Mode، وذلك بذكر الفئات اللازم استخدامها في كلا الوضعين، كما أتحدث عن طريقة ربطها بالأدوات وتكامل تقنية ADO.NET مع XML. **مقدمة لبرمجة ASP.NET** لأعرض فيه فلسفة عمل صفحات الخادم ASP.NET وبناء مواقعك المستخدمة لهذه التقنية، كما اعرض لك مجموعة من الكائنات التي لا غنى عنها في اغلب مشاريع ASP.NET، واختم الكتاب بالتحدث عن خدمات ويب Web Services وطريقة إنجازها واستخدامها.

القرص المدمج CD-ROM

يحتوي القرص المدمج المرفق مع هذا الكتاب على جميع الشيفرات المصدرية التي أضيف الرمز  في أعلاها، مقسمة الى مجموعة من المجلدات تمثل رقم الفصل الذي سطرت في الشيفرة. كما تعمدت إضافة ملفات صور (من النوع JPG) تمثل الصور والأشكال التوضيحية المعروضة في الفصول، وذلك خشية عدم وضوحها بين صفحات هذا الكتاب. كما يمكنك إيجاد الملف ArabicConsole.DLL والذي طورته لمحاكاة الكائن Console في الدليل الجذري للقرص - سأخبرك به لاحقا في الفصل الأول **تعرف على Visual Basic .NET**.

صفحة الكتاب على الانترنت

لا أود إنهاء علاقتي معك مع الصفحة الأخيرة من هذا الكتاب، بل سيكون شرف لي تمديدتها إلى فترات أطول. يمكنك عزيز القارئ اصطيادي على شبكة الانترنت إما عن طريق مراسلتي على بريدي الإلكتروني أو من خلال زيارة صفحة الكتاب إن كنت ترغب في الحصول على آخر التحديثات وتصحيح الأخطاء المتعلقة به، وذلك بتوجيه متصفحك إلى العنوان التالي:

<http://www.dev4arabs.com/lib/vbnetbook>

الأخطاء (هام جدا)

الوصول إلى الكمال شيء لا يتم إلا بالقدرة الإلهية، والعمل الذي أمام عينك جهد بشري متواضع ومعرض بنسبة كبيرة للخطأ، يهمني جدا التقليل من الأخطاء في هذا الكتاب، حتى يكون مرجعا عربيا سليما من الشوائب. في حالة العثور على أي خطأ (سواء لغوي أو تقني)، برجاء إبلاغي فورا عن الخطأ ورقم الصفحة الذي وقع فيها، وبهذا تسدي لي خدمة سأكون شاكرا ومقدرا لها.

كلمة أخيرة

أخي المبرمج العربي من المحيط إلى الخليج، بودي توضيح نقطة ضرورية تتمحور حول مؤلف الكتاب الذي تقرأه الآن. صحيح ان فن الكتابة بعيد كل المبعد عن مجال عملي إلا انه عليك معرفة أنني مبرمج ولست كاتب، فلست من الذين لديهم أعمدة في الصحف والمجلات ويهوى كتابة المقالات، بل أقوم بكتابة العديد من المشاريع والبرامج لمختلف القطاعات. كما لدي الكثير من العلاقات بين عمالقة المبرمجين في أنحاء المعمورة وقد استفدت الكثير والشيء الكبير من احتكاكي معهم. واعتقد -ل أكاد اجزم- انك لم تقتني هذا الكتاب لتتعلم فن البلاغة أو الاستمتاع بالتعابير اللغوية، بل تريد أن تتعلم لغة برمجة اسمها .NET. Visual Basic. لذلك، حاول قدر الامكان

تجاهل تعابيري اللغوية الركيكة، وضعف أسلوبي البلاغي والكتابي، ولنجعل لغة الشيفرات المصدرية Source Codes هي القاسم المشترك بيننا.

الفترة السابقة التي قضيتها في كتابة هذا الكتاب كانت طويلة بعض الشيء والوقت قصير جداً، ولم استطع -بصراحة شديدة- كتابة كل التفاصيل التي وددت ذكرها في هذا الكتاب، لذلك وضعت في عين اعتباري قدرة القارئ على التعلم الذاتي، حيث كان هدفي إعطائك مفتاح ومدخل لمسائل عديدة وتركت الباقي عليك للبحث عن التفاصيل الأخرى سواء في مواقع الانترنت او مستندات ومراجع NET Documentation الرسمية.

برمجة إطار عمل NET. باستخدام Visual Basic .NET ما هو إلا محاولة جادة لتأليف كتاب عربي من مؤلف عربي وموجه إلى مبرمج عربي لتقديم كل ما يحتاجه من معلومات تمكنه من بناء تطبيقات وحلول عملية موجه إلى إطار عمل NET Framework. باستخدام لغة البرمجة Visual Basic .NET، بدءاً من توضيح أساسيات لغة البرمجة وانتهاءً بتطوير تطبيقات مختلفة المجالات داعمة للغة العربية. أتمنى أن أكون قد وفقت في محاولتي هذه وقدمت ما يرضي ذوق المبرمج العربي ... حظاً سعيداً!

-- تركي العسيري

الظهران - فبراير 2003

turki@dev4arabs.com

الجزء الأول

الأساسيات

تعرف على Visual Basic .NET

بسم الله نبداً وعلى بركته نسير مع الجملة Visual Basic .NET، تتكون هذه الجملة من 14 حرفاً ونقطة واحدة، الحروف الـ 11 الأولى تعني لغة برمجة اسمها Visual Basic، والنقطة والحروف الثلاث الأخيرة تعني إطار عمل NET Framework.. لذلك، يمكننا ان نطلق على هذه اللغة Visual Basic for .NET Framework (أي لغة البرمجة Visual Basic الموجهة إلى إطار عمل NET Framework).

رحلة الألف ميل تبدأ بخطوة، وخطوتنا الأولى في رحلتنا مع Visual Basic .NET ستكون نظرية بحتة. فقبل أن تبدأ بفرقة أناملك لكتابة الشفرات وبناء برامجك، من الجيد اخذ فكرة مبسطة عن الحروف الشهيرة NET. ومعرفة ماذا تعني هذه الحروف، وما الذي تقدمه لك، وما هي الحاجة التي تدعونا -أنت وأنا على الأقل- للانتقال إلى برمجة إطار عمل NET Framework. وإذا كنت من المبرمجين المخضرمين، فسيكون هذا الفصل مصدر لإفراغ جميع معاناتك السابقة مع عالم البرمجة، أما إن كان هذا الكتاب أول كتاب برمجة تقرأه في حياتك، فاعتبر نفسك مبرمج محظوظ جداً لما ستكتشفه من التعقيدات التي كانت تواجه المبرمجين قبل تقنية ..NET

الحياة قبل NET.

لست هنا بصدد تأليف كتاب تاريخ أو التحدث عن بدايات ظهور الحاسبات الشخصية، ففي ذلك الزمن لم أكن شيئاً مذكوراً. ولكن ما أنا بصدده الآن هو تقديم عرض مقتضب وسريع لأساليب بناء البرامج منذ نظام التشغيل DOS وحتى لحظة كتابة هذه السطور، وسيكون حديثي موجهاً لمبرمجي نظم Microsoft بشكل حصري.

البرمجة تحت نظم DOS

كان كل ما هو مطلوب منك -كمبرمج- استخدام أمر Input لقنص المدخلات من المستخدم، والأمر Print لعرض المخرجات على الشاشة، بالإضافة إلى استخدام مجموعة من العبارات لتطبيق العمليات الحسابية. كانت في الحقيقة برمجة سهلة وممتعة للمبرمجين، حتى أصبح كل من هب ودب يدعي انه مبرمج، إلا أن النتيجة كانت برامج متشابهة، لا جديد فيها ولا تستخدم تقنيات جديدة.

بعد ذلك، ظهرت الحاجة إلى التحليق إلى مدى أبعد من الأسلوب السابق، فكانت أهداف التحليق -بشكل مبني- التفاعل مع الأجهزة Devices التي تتركب في الجهاز (كالطابعة، بطاقة الصوت، الفأرة... الخ)، إلا أن أجنحة المبرمجين في ذلك الوقت كانت تعتمد اعتماداً كلياً على برمجيات تابعة تسمى **المشغلات Drivers**. معظم هذه المشغلات كانت تتجزأ بلغة التجميع Assembly، وتتطلب خبرة كبيرة في التعامل مع المعالج وعتاد الكمبيوتر. فلو قمت بعمل برنامج يطبع النتائج على ورق الطابعة، فعليك إرفاق مشغل الطابعة مع البرنامج، وإذا أردت من برنامجك أن يعزف ملف صوتي، عليك إرفاق مشغل الصوت.

قد تبدو فكرة إرفاق ملفات المشغلات مقبولة -إلى حد ما- لبعض المبرمجين، إلا أن المشكلة الحقيقية التي كنا نواجهها هي أن لكل طابعة مشغل خاص بها. وبما أنه ليس لدينا أي فكرة عن نوعية الطابعة التي ستكون على طاولة المستخدم، فإن ذلك يفرض علينا إرفاق مشغلات لجميع أنواع الطابعات الموجودة في السوق. فلو تذكر برنامج Lotus 123 الذي يعمل تحت نظام MS-DOS، ستعلم أن البرنامج يرفق معه أكثر من 200 ملف، هذه الملفات ما هي إلا مشغلات لمختلف أنواع الطابعات. بالنسبة لي، كنت أفضل استخدام يدي لكتابة النتائج على الورق بدلاً من تحمل تكاليف نسخ الأقراص (كانت غالية جداً في ذلك الوقت) لوضع مشغلات الطابعات عليها.

الانتقال إلى Windows

أما مع نظام التشغيل Windows فقد حلت المشكلة السابقة، بحيث يتكفل نظام التشغيل بمهمة التعرف على عتاد الكمبيوتر وإرفاق مشغلاتها، فهو يوفر لك إمكانية الطابعة في برنامجك دون الحاجة لمعرفة نوعية الطابعة، ويمكنك من استخدام الصور والرسوم أو عزف ملفات الصوت أو استخدام الفأرة في برنامجك دون الالتزام بإرفاق مشغلات الأجهزة، أي كل ما هو مطلوب منك -كمبرمج- التركيز على برنامجك وصرف النظر عن الأمور التقنية الدنيا كالأجهزة والعتاد، إدارة الذاكرة، إدارة الأقراص وغيرها، والتي يتكفل بها نظام التشغيل بكل اقتدار.

إلا أن البرمجة تحت بيئة Windows تختلف اختلافاً جذرياً عن البرمجة تحت بيئة DOS، فبرنامجك لم يعد يستخدم الطرق التقليدية لقنص المدخلات وعرض المخرجات، فقنص المدخلات أصبحت تتم من قبل نظام التشغيل، والذي يقوم بإرسالها لك على شكل رسائل **Messages** كالنقر Click، الضغط على زر KeyDown... الخ. لذلك، انقلبت الموازين البرمجية في حياة معظم المبرمجين، لتصبح برامجهم تحتوي على عشرات -بل مئات- الحلقات التكرارية لقنص هذه الرسائل.

أما من ناحية عرض المخرجات، فلم يعد هناك شيء اسمه Print لإظهار المخرجات على الشاشة، حيث يتطلب نظام التشغيل Windows من المبرمجين إنشاء نوافذ وسياقات رسم وتسجيل طبقات ليتمكنوا من عرض المخرجات من خلالها. فلو أراد مبرمج تعلم لغة برمجة جديدة لكتابة أول برنامج شهير Hello World تحت بيئة Windows، عليه كتابة عشرات السطور المعقدة جداً لعمل ذلك، الأمر الذي أدى تقاعد المبرمجين من عالم البرمجة وفضلوا العمل عند شركات سيارات الأجرة (التاكسي)! وحتى أريك الأمر الواقع، انظر إلى هذه الشيفرة المعدة بلغة C والتي تمثل برنامج يقوم بعرض نافذة خالية فقط:

```
#include <windows.h>

LRESULT CALLBACK MainWndProc( HWND, UINT, WPARAM, LPARAM );

HINSTANCE ghInstance;

int PASCAL WinMain( HINSTANCE hInstance,
HINSTANCE hPrevInstance,
LPSTR lpszCmdLine,
int nCmdShow )
{
    WNDCLASS wc;
    MSG msg;
    HWND hWnd;

    if( !hPrevInstance )
    {
        wc.lpszClassName = "ShowWindow";
        wc.lpfnWndProc = MainWndProc;
        wc.style = CS_OWNDC | CS_VREDRAW | CS_HREDRAW;
        wc.hInstance = hInstance;
        wc.hIcon = LoadIcon( NULL, IDI_APPLICATION );
        wc.hCursor = LoadCursor( NULL, IDC_ARROW );
        wc.hbrBackground = (HBRUSH)( COLOR_WINDOW+1 );
        wc.cbClsExtra = 0;
        wc.cbWndExtra = 0;
```

```

        RegisterClass( &wc );
    }

    ghInstance = hInstance;

    hWnd = CreateWindow ( "ShowWindow",
        "ShowWindow",
        WS_OVERLAPPEDWINDOW|WS_HSCROLL|WS_VSCROLL,
        0,
        0,
        600,
        300,
        NULL,
        NULL,
        hInstance,
        NULL);

    ShowWindow( hWnd, nCmdShow );

    while( GetMessage( &msg, NULL, 0, 0 )) {
        TranslateMessage( &msg );
        DispatchMessage( &msg );
    }

    return (int) msg.wParam;
}
LRESULT CALLBACK MainWndProc( HWND hWnd, UINT msg, WPARAM wParam,
    LPARAM lParam )
{
    PAINTSTRUCT ps;
    HDC hDC;

    switch( msg ) {

    case WM_DESTROY:
        PostQuitMessage( 0 );
        Break;

    default:
        return( DefWindowProc( hWnd, msg, wParam, lParam ));
    }

    return 0;
}

```

ولو أنهم صبروا لكان خيرا لهم، فبعد فترة ليست طويلة ظهرت حلول من كبريات شركات صناعة البرمجيات لتسهيل عملية البرمجة تحت نظم Windows، وذلك باختراع الكلمة السحرية **Visual**، فكل ما هو مطلوب من المبرمج تصميم شاشات (نوافذ) برنامجه بالفأرة، وكتابة بضعة

أو امر يتم تنفيذها بمجرد قيام المستخدم بالتفاعل مع برنامجه سواء بالفأرة أو لوحة المفاتيح أو انفه (لقد شاهدت فعلاً احد الزملاء يستخدم انفه على الشاشة!).

بعد ذلك، لاحظ المبرمجون أن نسبة كبيرة من شيفرات برامجهم مكررة وقد كتبت في عشرات المشاريع، فلو أمعنت النظر قليلاً، لوجدت ان معظم تطبيقات Windows تشارك إلى حد كبير في معظم وظائفها الشائعة، لذلك كان على مطوري نظام Windows أيجاد حلول لتبادل البيانات ومشاركة الشيفرات بين البرامج، إلا أن تحقيق هذا الهدف بدا مستحيلاً لبعض الوقت، لان جميع برامج Windows تعمل في مناطق مختلفة ومستقلة بها في الذاكرة تسمى **مساحات العنوان Address Spaces**، لذلك أسس مطورو Windows أسلوباً أو بروتوكول برمجي يسمح للتطبيقات بالتخاطب فيما بينها بمعايير ومواصفات قياسية يسمى **التبادل الديناميكي للبيانات Dynamic Data Exchange - (DDE)**.

إلا أن DDE كانت بها الكثير من العيوب التي حثت بالمبرمجين إلى تجنب استخدامها، كثرة الانتهيارات التي تحدث في البرامج، والاتصالات دائمة الانقطاع بين التطبيقات، بالإضافة إلى صعوبة وتعقيدات الشيفرة المصدرية وغيرها، إلى أن قامت Microsoft بإصدار تقنية **ربط الكائنات وتضمينها Object Linking & Embedding - (OLE)** والتي تعتمد في بنيتها التحتية على DDE، حيث وفرت قابلية لتبادل البيانات بين البرامج والتطبيقات المختلفة لتمكنك - مثلاً- من إدراج جدول من Microsoft Excel لتضمينه أو ربطه في مستند Microsoft Word.

في أواخر عام 1993 غيرت Microsoft البنية التحتية لـ OLE حيث لم تعد تعتمد على DDE وتم إعادة بنائها من جديد لتصدر ما سمي **OLE2**، والتي مكنت المبرمجين من تطبيق أسلوب **العمل في نفس المكان In-place Activation** بحيث يمكنك تحرير جدول Excel وأنت بداخل مستند Word في نفس النافذة ودون الحاجة لمغادرة Word.

الحلم أصبح حقيقة مع COM

من الإنجازات التي أحدثت ثورة كبيرة في عالم تطوير البرامج تحت Windows، تقنية **برمجة المكونات Component Object Model - (COM)**، حيث مكنت هذه التقنية المبرمجين - بلغات البرمجة المختلفة- من المشاركة في تطبيقاتهم بأسلوب **كائني التوجه Object Oriented**. ليس هذا فقط، بل تعدى الأمر أكثر من ذلك ليصل إلى **المكونات الموزعة Distributed COM - (DCOM)**، لتصبح مكونات البرامج موزعة على أجهزة مختلفة، ويتم تبادل البيانات عن طريق شبكة الانترنت بشكل مذهل، فيستطيع صديقي في منغوليا من

استخدام بعض أجزاء برنامجي الموجود في جهازي المحمول الذي اصطحبه معي في رحلاتي البرية بصحراء الربع الخالي.

أثرت COM بشكل إيجابي كبير في عالم تطوير البرامج تحت Windows، لدرجة ظهور شركات متخصصة فقط في تطوير مكونات COM (كأدوات التحكم ActiveX Controls، مكتبات فئات DLL ActiveX.... الخ)، وأصبحت عملية بناء البرامج تعتمد على البرمجة مكونية التوجه **Component Oriented Programming** بشكل كبير، ولا تكاد تجد أي برنامج الآن إلا ويستخدم مكونات COM.

مع ذلك، فإن استيعاب البنية التحتية لبرمجة المكونات COM مسألة صعبة جداً، فهي تتطلب التوغل في تفاصيل معقدة لاستخدام ما يسمى **الواجهات Interfaces**، وكثرة الأخطاء والشوائب البرمجية أصبحت أمراً طبيعياً، وعند الحديث عن مصادر النظام System Resources فحدث ولا حرج، فهي تستهلك الكثير من المساحات الغير مستخدمة لعدم تفرغ أجزاء الذاكرة من الكائنات المنشأة، إما بسبب الانهيارات المفاجئة للبرامج، أو نسيان حذف مؤشرات الكائنات التي انشأها أو استخدمها البرنامج.

من ناحية أخرى، فإن مكونات COM تعتمد اعتماداً كلياً على **مسجل النظام Windows Registry**، وأي مشكلة تحدث في هذا المسجل تؤثر على باقي المكونات المثبتة في الجهاز، ولن تستطيع استخدامها إلا بإعادة تركيب Reinstall البرامج التابعة لها. وعملية تركيب البرامج بحد ذاتها معقدة جداً، إذ تتطلب منك نسخ ملفات المكونات ومن ثم تسجيلها في المسجل وإعدادها والتحقق من الإصدارات الأقدم ومن ثم تعريفها على الشبكة المحلية (إن كانت DCOM)، وأي خطأ في عملية تثبيت البرامج، يؤدي إلى حدوث كارثة في جهاز المستخدم والتأثير على باقي البرامج المثبتة في الجهاز، ليكون الحل الوحيد إعادة تهيئة Format القرص الصلب. وعند الحديث عن التوافقية، فلا يمكنك استخدام إصدارين مختلفين لنفس المكون بسبب مشكلة تسمى **Versioning** ليس هذا مجال تفصيلها.

تحديات الانترنت

مع ظهور الانترنت أصبحت مسألة تكامل التطبيقات مع هذه الشبكة أمر ضروري إن لم يكن إلزامي، فعمليات تحديث البرامج وتبادل البيانات فيما بينها يمكن أن تخفف تكاليفها عن طريق الانترنت، أضف إلى ذلك مدى السهولة التي تمكن مستخدمين الكمبيوتر -حول العالم- من الوصول إلى المعلومات بالتصفح في المواقع المختلفة. هذه الشبكة لم يحصر مجالها في عرض

المواقع فقط، بل تعدى الأمر أكثر من ذلك إلى أن يصل لمستوى تطوير تطبيقات ويب باستخدام تقنيات معالجة الصفحات على أجهزة الخوادم كتقنية **ASP - Active Server Pages**. مع ذلك، فإن تطوير النظم الكبيرة باستخدام ASP أمر معقد جداً، ويتطلب عشرات التنقيحات والتعديلات للصفحة الواحدة، خاصة إن علمت أن برمجة ASP ليست كائناتية توجه OOP، حيث أنها كانت تعتمد على لغة البرمجة المصغرة VBScript لكتابة شيفرات الصفحات. إلا أن دعم صفحات ASP لبرمجة المكونات COM أدى إلى تكامل حقيقي وتسهيل أكبر ومرونة أكثر في تطوير النظم -خاصة الكبيرة، ليتمكن المستخدمون من تطوير تطبيقات متعددة الطبقات nTied Applications حقيقية، تعتمد على كائنات ADO كخلفية النظام لإدارة قواعد البيانات، ومكونات COM لأداء وظائف النظام كطبقة وسطي، وصفحات ASP لتمثل واجهة المستخدمين.

انتشرت نظم ASP انتشاراً كبيراً بين المواقع العالمية، إلا أن المشكلة التي واجهت مطوري نظم ويب تظهر عند تبادل وتشارك البيانات بين المواقع المختلفة، فمثلاً لو كنت تصمم موقع لبيع المادة الخام المستخدمة في إنتاج الفلافل (الطعمية) وأردت من هذا النظام أن يستقبل الطلبات من مختلف الدول العربية، وعلى الأرجح ستنم عملية الشراء بالعملة المحلية للدولة، أضف إلى ذلك مسألة التسعير حيث أنها تعتمد اعتماداً كلياً على القوة الشرائية لعملة تلك الدولة، وسوق المطاعم المنافسة في إنتاج سندويشات الفلافل، لذلك قد تحتاج إلى تبادل البيانات مع مواقع أخرى تحدد لك السعر المناسب في وقت معين، ومواقع أخرى تمكن المستخدمين من مراقبة عمليات شحن الفلافل وبيان مواقعها الحالية.

إن أردت تبادل البيانات مع هذه المواقع بشكل تقليدي، فالعملية تتم بإرسال صفحات HTML من موقع إلى آخر، عليك القيام بجميع الخطوات اللازمة لحذف الوسوم (Tags) (التي تستخدم لتنسيق صفحات HTML)، وهذه بحد ذاتها عملية معقدة جداً، وتتطلب جهد ووقت إضافي. ليس هذا فقط، بل أن أي تعديل بسيط في صفحات أحد المواقع قد يؤدي إلى كارثة في عملية نقل البيانات وعدم دقتها وقد يصل الأمر إلى أن تكون الطلبية فلافل ويتم شحن سندويشات كبدية بلدي!

لذلك اعتمد المطورون على تقنية **(SOAP) - Simple Object Access Protocol** بحيث يتم نقل البيانات عن طريق لغة XML (وهي لغة وصف البيانات وليس تنسيق البيانات كـ HTML) واستخدام بروتوكولات الاتصال TCP/IP.

قد تكون فكرة SOAP ممتعة جداً، ولكن -مع الأسف الشديد- استخدام هذه التقنية معقد جداً، ويتطلب الكثير من الخبرة البرمجية، بحيث انحصار استخدامها على المبرمجين المحترفين

فقط، كذلك كثرة الأخطاء والشوائب البرمجية Bugs في تطوير نظم تعتمد على SOAP أمر لا مفر منه.

عشرات التقنيات لأداء الوظائف

كما رأيت في الفقرات السابقة، فإن تطوير البرامج مسألة معقدة جدا وتتطلب دراية كافية في التعامل مع التقنيات المختلفة، فلكي تطور مواقع ويب ديناميكية عليك تعلم VBScript (إن كانت من جهة العميل) وتعلم ASP (إن كانت من جهة الخادم)، وإن أردت بناء نظم قواعد بيانات عملاقة عليك إتقان لغات الاستعلام المتقدمة كـ T-SQL للحصول على أكبر قدر من تحسين للكفاءة Optimization، وإن أردت تطوير مكونات COM بفاعلية أكثر ودون حدود عليك تعلم أحد لغات البرمجة المتقدمة كـ Visual C++، وإن أردت مخاطبة تطبيقات Microsoft Office الشهيرة فلا مخرج لك إلا باستخدام VBA، أما إن أردت تطوير برامج تعمل تحت نظم Windows بسهولة وكسر حاجز الوقت فستجد ضالتك في Visual Basic.

ليس هذا فقط، بل حتى الوظائف المتشابهة تنجز بتقنيات مختلفة، فلديك مثلاً التقنيات DAO، ADO، و RDO لتطوير التطبيقات المعتمدة على قواعد البيانات Databases، وهناك أيضاً مجموعة من التقنيات كـ GDI، DirectX، و OpenGL لتطوير النظم التي تعتمد على الصور والرسوم بكثرة.

كانت هذه جولة سريعة حول بناء البرامج في السنوات السابقة، تذكر أن كلامي موجه لتقنيات Microsoft فقط. وإن لم تكن لديك أي معرفة بما سبق ذكره من تقنيات برمجية، فأرجو أن لا يصيبك ذلك بالإحباط، بل على العكس من ذلك، يمكنك أن تعتبر نفسك مبرمج محظوظ جداً، حيث ستبدأ حياتك الجديدة من حيث انتهى الآخرون، ومع أحدث وأفضل تقنية تستخدم لتطوير التطبيقات في القرن الحادي والعشرين وهي Microsoft .NET.

الحياة بعد .NET

لا ادعي هنا علم الغيب ووصف مستقبل تطوير التطبيقات، ولكن يمكنك اعتبار ما سأذكره في الفقرات التالية عرض سريع لاستراتيجية .NET. وبيان مدى تأثيرها على صناعة البرمجيات.

الاستقلالية عن منصات العمل

اكتب البرنامج مرة واحدة فقط وسيتم تنفيذه على مختلف منصات العمل المختلفة: كالأجهزة المحمولة Notebooks، خادמות Servers، هواتف جواله Mobiles، تليفزيونات رقمية Digital TVs، ثلاجات، طائرات، أبواب كراج، سيارات، وكل شيء رقمي Digital. وإن كنت شخص تسكن في منطقة بعيدة عن عائلتك ولا تجيد الطبخ، فيمكنك طلب شيفرة مصدريّة من الوالدة لكتابة برنامج لتحضير الكبة ومن ثم تركيبه في الفرن (الذي سيكون رقمي لاحقاً) لإنجاز الكبة. وهذا بفضل استقلالية برامجك عن منصات العمل الذي تقدمه .NET.

الاستقلالية عن منصات العمل لا تنحصر حول العتاد Hardware فقط، بل تشمل نظم التشغيل المختلفة، فحالياً برامجك يمكنها أن تعمل على مختلف إصدارات نظام التشغيل Windows، وقريباً قد نرى أن إطار عمل .NET Framework س يدعم في أنظمة التشغيل الأخرى كـ Linux® وحتى Macintosh®.

النتيجة الإيجابية من استقلالية برامجك عن منصات العمل تقتضي منك -كمبرمج- التركيز على برامجك فقط وصرف النظر عن العالم الخارجي أو المكان الذي سيتم تنفيذ البرنامج فيه، مع ذلك قد تسأل سؤال بديهي ونقول: إن كانت البرامج مستقلة عن منصات العمل، فكيف سيتم تنفيذ الوظائف المختلفة والتي لا تتوفر في منصة عمل معينة، والجواب بكل بساطة يعتمد على نوعية البرنامج الذي تصممه، فبكل تأكيد الشيفرة المصدريّة لبرنامج الوالدة (تحضير الكبة) لن تقوم بتشغيله في غسالة الملابس (التي ستكون رقمية أيضاً)، وإنما موجهه إلى الفرن الرقمي. لذلك أريد توضيح أن المقصد من قضية استقلالية البرنامج عن منصات العمل ميزة من إطار عمل .NET Framework وليس للمبرمج أي علاقة مباشرة بها، فكل ما هو مطلوب منه كتابة البرنامج فقط بحيث يلائم البيئة التي سيعمل بها.

.NET نسخة محسنة من COM

قد تُفاجأ إن أخبرتك أن الاسم الابتدائي لمشروع .NET كان يسمى COM 2.0، أي الجيل التالي من برمجة المكونات COM، وهذه بحد ذاتها حقيقة إن أخذتها بشكل نظري. فالفكرة من COM و .NET تقريباً متطابقة -نظرياً- من منطلق توزيع الشيفرات والاستقلالية الشبه تامة عن منصات العمل، إلا أن .NET تختلف اختلافاً جوهرياً كبيراً في بنيتها التحتية عن COM، حيث أن تقنية .NET تم إعادة بنائها من جديد وعولجت العشرات من المشاكل التي واجهت ميرمجي COM سابقاً.

أول مشكلة ابتدائية تم حلها هي الاستغناء عن مسجل النظام System Registry، حيث أن مكونات NET. تصل إليها وتستعلم عنها مباشرة عن طريق ملفاتها، دون الحاجة إلى المرور بمسجل النظام كما كنا نفعل سابقاً مع COM، وهذا يعني أن عملية تثبيت البرامج لا تتطلب جهد إضافي لإنجازها، فيكفي نسخ الملفات من القرص المدمج إلى القرص الصلب وسيعمل البرنامج دون أية مشاكل، مع ذلك قد تحتاج إلى برامج التركيب لتنفيذ بعض اللمسات الخفيفة (كوضع الملفات في أماكنها المناسبة، تخصيص العناصر المطلوب تثبيتها، اعدادات بسيطة قليل عملية تنفيذ البرنامج.... الخ).

وبالنسبة للمكونات الموزعة DCOM فلن تحتاج إلى العبث في نظام التشغيل Windows ومحتويات المكون لتجري عشرات الاعدادات الإضافية حتى يتم توزيعه، إذ أن مكونات NET. هي موزعة بحد ذاتها.

أما مشكلة التوافقية Versioning فلن تحدث بعد الآن، حيث يمكن تثبيت إصدارين مختلفين من نفس المكون دون أن يؤثر أحدهما على الآخر.

ميزة عظيمة أخرى في مكونات NET. لم تكن مدعومة سابقاً مع مكونات COM وهي الوراثة Inheritance، فمكونات COM لم يكن متاحاً اشتقاقها وراثياً وتطوير فئاتها، أما مكونات NET. فلديك القدرة الكاملة لاشتقاق فئات المكونات وراثياً دون الحاجة للحصول على شيفراتها المصدرية.

انظر أيضاً

الوراثة والاشتقاق الوراثي مواضيع دسمة، سأحدث عنها لاحقاً في الفصل الرابع الوراثة .

صحيح أن مكونات COM كانت تزيل حاجز الفروقات بين لغات البرمجة المختلفة، إلا أن هذا الحاجز لم يتم إزالته بشكل كامل، فما زال مبرمجو بعض لغات البرمجة (كـ Visual Basic) يواجهون مشاكل وصعوبات في استخدام بعض مكونات COM المنجزة بلغات متقدمة أخرى (كـ Visual C++) خاصة مع المكونات التي تتعامل مع أنواع بيانات ليست مدعومة في Visual Basic (كالمؤشرات Pointers مثلاً)، ولكن مع مكونات NET. أمست كل هذه التعارضات من الماضي، ومرد ذلك ان جميع لغات NET. موحدة بفضل معايير CRL كما ستري لاحقاً.

تكامُل لغات البرمجة

جميع لغات .NET متكاملة فيما بينها، فبرنامج المصمم بـ Visual Basic .NET يمكن إضافة بعض العناصر والشيفرات المصدرية إليه من لغة Visual C# .NET دون أي مشاكل، بل يمكن للمشروع الواحد أن يدمج شيفرات مصدريّة من لغات متعددة مثل: .NET، Delphi، Java، .NET، Visual C++، .NET، Turki (إن وجدت)، .NET، Fortran... الخ وذلك بفضل معايير CRL التي توحد لغات البرمجة.

قد تتساءل وتقول، ما دامت لغات البرمجة المختلفة موحدة بهذا الشكل فما الفائدة من تعلم أكثر من لغة؟ والجواب هو أنه ما زالت كل لغة برمجة تحتوي على سمات ومميزات خاصة بها، وإعني بكلمة خاصة بها في هذا السياق هو عدم إمكانية تكاملها مع لغات .NET. الأخرى أن تم تفعيل هذه المزايا. من ناحية أخرى، جميع لغات .NET. يتم تحويلها إلى لغة MSIL لحظة الترجمة Compiling كما ستري لاحقاً.

خدمات ويب هي مستقبل الانترنت

منذ أن نشبك سلك الهاتف وتصل بالانترنت، فإن معظم وقتنا نقضيه على المتصفح Browser للوصول إلى المواقع المختلفة، الفعل السابق سيكون من أساطير الأولين في المستقبل القريب، وذلك بعد انتشار خدمات ويب Web Services حيث ستغير الكثير من أسلوب تعاملنا مع شبكة الانترنت بطرق لم تخطر على بال انس ولا جان. الجزء الخامس برمجة ويب من هذا الكتاب يقدم لك شرحاً وافياً حول خدمات ويب، ولكن دعني هنا أعرف لك ما هي خدمة ويب Web Service بشكل سريع.

خدمة ويب (تسمى أحياناً XML Web Service) ما هي إلا برنامج يستقبل طلبات Requests ومن ثم يستجيب لها Response باستخدام بروتوكول HTTP تحت معايير لغة XML القياسية، وبذلك تتمكن ملايين المواقع المنتشرة حول العالم من تبادل البيانات فيما بينها وإنجاز الأعمال المطلوبة. يمكنك اعتبار خدمات ويب على أنها برنامج يتصل بالعالم الخارجي عن طريق شبكة الانترنت مما يمكنك من إرسال واستقبال البيانات بصيغة XML دون أن يتطلب الأمر منك أي خبرة في لغة XML أو بروتوكولات TCP/IP، حيث يوفر لك إطار عمل .NET Framework كل ما تحتاجه لتطوير وبناء خدمات ويب.

ستنتشر خدمات ويب انتشاراً لا مثيل له في مواقع الانترنت، (سنوفر خدمة ويب في موقع شبكة المطورون العرب بمشيئة الله)، بل ستصل إلى تغيير معظم الأنظمة الحالية لتعتمد على خدمات ويب وتصبح مؤجرة، فيمكنك مثلاً توفير خدمة ويب للتذكير بالمواعيد، أو خدمة توفير

أسعار العملات التي تحدث على مدار الساعة، أو خدمة الحصول على دروس ومقالات من موقع آخر، أو خدمة بحث كخدمة Goggle Search... الخ.

ماذا عن المبرمج العربي؟

إطار عمل NET Framework. موجه إلى جميع المبرمجين حول العالم باختلاف لغاتهم الطبيعية، واللغة العربية إحدى هذه اللغات، فالنصوص مثلًا جميعها تتبع الترميز UNICODE وتوزيع محارف لوحات المفاتيح العربية مأخوذ في عين الاعتبار، كذلك الحال مع مواصفات البيئة كتسويق العملة، الوقت والتاريخ، الأرقام، نظام الفرز... الخ، فهي مدعومة لجميع الدول العربية باستثناء -مع الأسف الشديد- السودان، فلسطين، جيبوتي، الصومال وموريتانيا فلم اجد لها في احد روابط مستندات NET Documentation. الرسمية.

وعند الحديث عن المسائل التقنية الأخرى، فتقنية **المرآة Mirroring** مدعومة بشكل جيد في نماذج Windows Forms (كما سترى في الجزء الثالث **تطوير تطبيقات Windows**)، وبالنسبة للتاريخ الهجري فيوفر لك إطار عمل NET Framework. الفئة HijriCalendarClass التي يمكنك من استخدام التاريخ الهجري في برامجك وتوفر لك طرق وخصائص لتعديل وضبط القيمة الصحيحة لليوم والشهر.

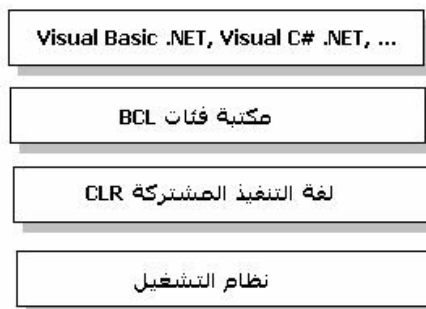
انظر أيضا

سترى تطبيقات وأمثلة حول استخدام الفئة HijriCalendarClass في
الفصل السادس **الفئات الأساسية**.

باختصار، وفرت Microsoft منصة تطوير قوية تدعم اللغة العربية بشكل رائع، لتتخصص المسؤولية علينا نحن كعرب سواء في شركات عربية أو مطورون عرب - لتقديم كافة الحلول الفعالة للمستخدم العربي.

محتويات إطار العمل .NET Framework

والآن سيتمحور حديثي حول معمارية ومحتويات إطار عمل .NET Framework، يمكنني ان اقسم لك محتوياته إلى أكثر من 10 طبقات، ولكنني فضلت تقليص العدد -للتسهيل عليك- كما ترى في (الشكل 1-1):



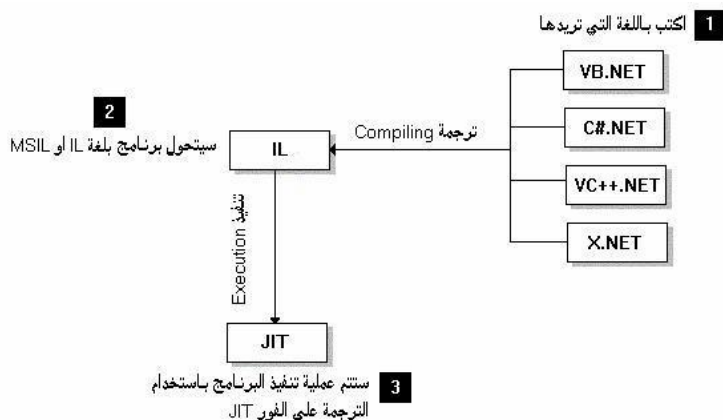
شكل 1-1: محتويات إطار عمل .NET Framework

مكتبة فئات Base Class Library (تسمى أيضا مكتبة فئات .NET Framework Class Library) هي عبارة عن مئات الفئات الموجودة في عشرات ملفات DLL تعتبر كنزاً غالباً يسيل له لعاب المبرمجين الجادين، حيث تحتوي كل ما تحتاجه لإنجاز برامجك ومشاريعك بدءاً بتقديم فئات لوصف البيانات الأساسية (كـ Integer، String) إلى إدارة خرج ودخل الملفات I/O File Processing، مسارات التنفيذ Threading، الصور والرسوم، نماذج Windows Forms، نماذج Web Forms، الاتصال بقواعد البيانات ADO .NET وغيرها الكثير. بالنسبة للغة التنفيذ المشتركة **Common Language Runtime - CLR** فهي موحدة لمعايير جميع لغات .NET. الأخرى، كما أنها المسؤولة عن عمليات إدارة الذاكرة Memory Management، تفريغ مصادر النظام باستخدام Garbage Collection، أخطاء وقت التنفيذ Exception Handling (سأتحدث عن كل ما ذكرته في الفصول اللاحقة من هذا الكتاب).

أخيراً، نظام التشغيل الذي يمكنك من تثبيت إطار عمل .NET Framework عليه هو Windows فقط (لحظة كتابة هذه السطور)، ولكن قد ترى في القريب العاجل نظم تشغيل أخرى داعمة له.

الترجمة على الفور JIT

من أجمل الأشياء التي ستكتشفها والتي تعتبر فتح كبير -في رأيي الشخصي- في عالم برمجة .NET هو أسلوب الترجمة على الفور **Just In Time Compiling - (JIT)**، وهي تقنية تقوم بترجمة البرنامج عند تنفيذه حيث ينتج أفضل شيفرة تتناسب مع الجهاز الذي سيعمل عليه البرنامج مما ينتج عنه نتائج إيجابية جيدة جدا (هذا عند الحديث عن تحسين الكفاءة Optimization) وحتى تعلم كيف يحدث ذلك تابع (الشكل 1-2):



شكل 1-2: مراحل ترجمة وتنفيذ البرنامج.

اختر أي لغة تناسب مزاج أناملك واكتب الشيفرة بها، وعند قيامك بعملية الترجمة سيتم تحويل ملف البرنامج إلى ملف شبيه بالملفات التنفيذية Executable File مكتوب بلغة جديدة اسمها **Microsoft Intermediate Language (تختصر IL أو MSIL)**. حيث تحتوي على شيفرات البرنامج ولكنها غير قابلة للتنفيذ مباشرة، بل يشترط وجود مترجم على الفور JIT Compiler حيث يقوم بترجمة هذا الملف الثنائي إلى لغة الآلة معطيا أفضل التعليمات بحيث تناسب الجهاز الحالي، فعندما تصمم برنامجك مرة واحدة فقط باستخدام Visual Basic .NET. فاعلم ان البرنامج سيستفيد من كل مصادر العتاد ونظام التشغيل الذي يتم تنفيذ البرنامج فيه، أي لو شغلت برنامج تحت خادم Server يحتوي على 9504378250470592 معالج فحق ثقة تامة ان برنامجك سيستفيد من كل هذه المعالجات رغم انك لم تهتم بهذه النقطة على الأرجح.

عملية الترجمة JIT تقوم بترجمة كل جزء من البرنامج عند الحاجة أي عند استدعاء وظيفة معينة فيه، ويمكنك ترجمة البرنامج من أوله إلى آخره أيضا عن طريق مترجم آخر يسمى (NGEN) – Native Image Generator (يسمى أيضا Pre-JIT Compiler) مع معرفة أن الترجمة تتم مرة واحدة فقط، ولن يشعر المستخدم بأي بطء في عملية تنفيذ البرنامج عند تشغيله مرة أخرى، فقد تمت ترجمته بالشكل المناسب لجهازه. (راجع مستندات .NET Documentation لمزيد من التفاصيل حول المترجم (NGEN)).

المجمعات Assemblies

المجمع Assembly ما هو إلا ملف (قد يكون EXE أو DLL) يحتوي على كل شيء يتعلق بالبرنامج سواء شيفراته المصدرية بعد الترجمة Compiling، الصور والرسوم، ملفات المصادر Resource Files، صفحات HTML، وغيرها. كل هذه العناصر يمكنك أن تضعها جميعا في ملف واحد فقط.

في اغلب الأحوال، يمثل المجمع برنامج واحد، مع ذلك يمكن للمجمعات Assemblies أن تحتوي على مجمعات أخرى، أي -بعبارة أخرى- يمكن لبرنامجك أن يدمج في داخله برنامج آخر -كما ستري لاحقا في الفصل الحادي عشر **المجمعات Assemblies**.

أخيرا، كل ما ذكرته من جمل وعبارات مبهمة في الفقرات السابقة، سأعود للحديث عنها بالتفصيل الممل في باقي فصول هذا الكتاب، فلست بحاجة لأن تكون مستوعبا لكل شيء الآن، إذ أن كل ما كنت أريده الآن هو تقديم جولة سريعة حول محتويات إطار عمل .NET Framework.

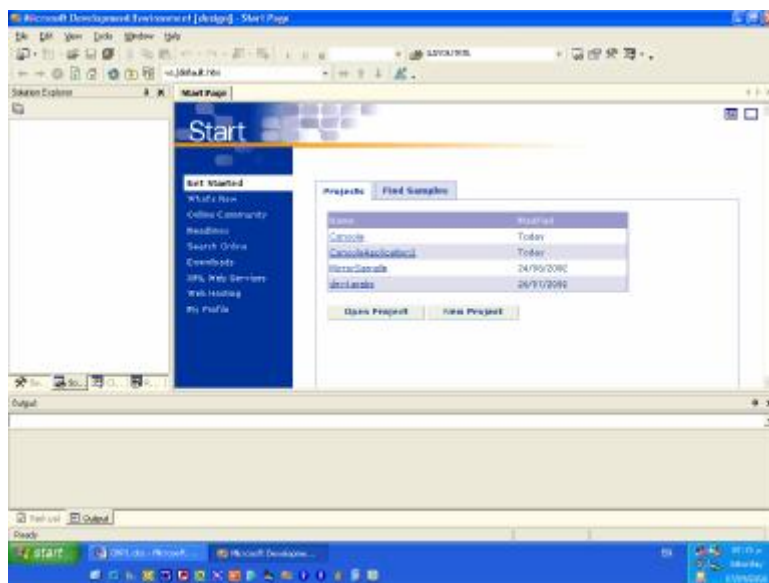
بيئة التطوير Visual Studio .NET

قد أغير العرف المتبع في كتب البرمجة (والتي تبدأ بشرح بيئة التطوير بالتفصيل)، ولكن من منطلق إيماني الشديد بأن الشخص -الذي يقرأ كتابي الآن- هو مبرمج وليس مستخدم، فاعتقد انه قد وصل إلى مرحلة تمكنه من تعليم نفسه ذاتيا لاستخدام بيئة التطوير. لذلك، التمس منك العذر الشديد يا عزيزي القارئ في مسالة شرح بيئة التطوير، فلن تجد هنا إلا عرض سريع ومختصر لبعض محتويات البيئة. مع ذلك، قد أنطرق في فصول لاحقة إلى تفصيل بعض النواظ متى ما دعت الحاجة لذلك.

تتشارك معظم لغات البرمجة (Visual Basic .NET، Visual C#.NET، Visual NET، C++ وغيرها) في بيئة تطوير متكاملة من Microsoft تسمى Visual Studio. NET، حيث توفر لك كل ما تحتاجه من خدمات وأدوات في قمة الروعة تسهل لك حياتك البرمجية. شرح جميع محتويات Visual Studio .NET يتطلب -دون مبالغة- كتاب كامل، وقد أصدرت بالفعل مطابع Microsoft Press كتابا يشرح كل صغيرة وكبيرة حول هذه البيئة، أما الكتاب الذي نقرأه فهو يتعلق بلغة البرمجة Visual Basic .NET فقط، ولن اذكر إلا النوافذ والأدوات التي ستستخدمها باختصار هنا.

نوافذ بيئة التطوير

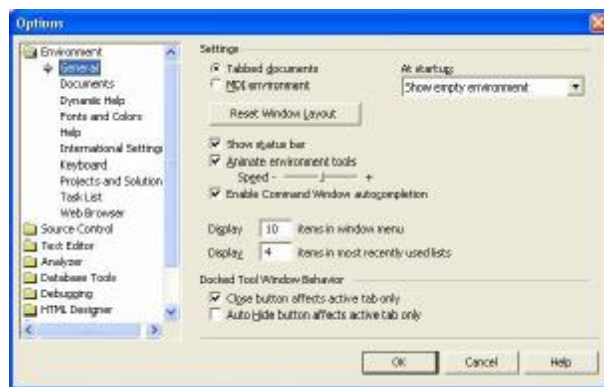
عند تشغيل بيئة Visual Studio .NET لأول مرة، مستظهر لك النافذة Start Page (شكل 1-3)، تستطيع الوصول إلى آخر مشاريع تم فتحها أو إنشاء مشاريع جديدة عن طريق هذه النافذة، كما انك تستطيع الوصول إلى آخر الأخبار المتعلقة بـ .NET Framework. بالضغط على الرابط Headlines وذلك لان النافذة Start Page ما هي إلا صفحة ويب تقليدية.



شكل 1-3: الشاشة الافتتاحية لبيئة التطوير Visual Studio .NET.

نافذة الخيارات Options:

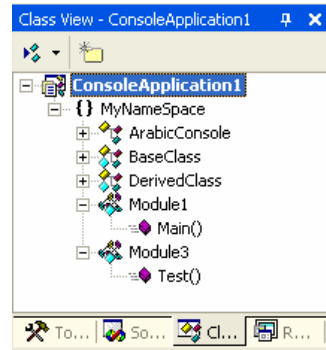
اختيارك للأمر Options من قائمة Tools يؤدي إلى ظهور هذه النافذة (الشكل 1-4)، حيث يمكنك من تخصيص وإعداد عشرات الأوضاع والخيارات كاعدادات بيئة التطوير، محرر الشيفرات، المترجمات Compilers، قواعد البيانات Database، التنقيح Debugging، محرر صفحات HTML، مصمم نماذج Windows Forms وغيرها.



شكل 1-4: نافذة الخيارات Options.

نافذة عرض الفئات Class View:

الغرض الرئيسي من هذه النافذة هو عرض فئات البرنامج Project Classes على شكل شجري (شكل 1-5)، مع العلم أن الفئات التي تعرض في هذه النافذة هي الفئات التي تعرفها في الشيفرة المصدرية للمشروع الحالي فقط، فلا نتوقع ظهور فئات من مكتبات أخرى لم يتم تضمينها في نفس المشروع.

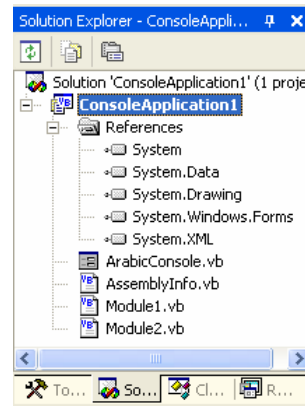


شكل 1-5: نافذة عرض الفئات Class View

الضغط المزدوج على عنصر من عناصر هذه الشجرة، يؤدي إلى فتح نافذة المحرر ونقل مؤشر الكتابة Cursor إلى منطقة تعريف الفئة أو العضو في الفئة. اختر الأمر Class View من قائمة View لعرض هذه النافذة.

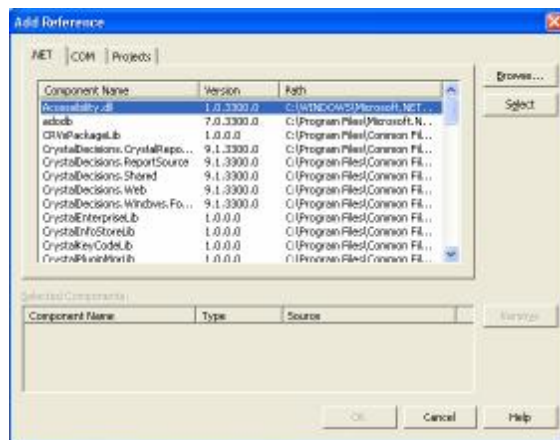
نافذة مستكشف الحل Solution Explorer:

إن كانت نافذة عرض الفئات السابقة تعرض فئات المشروع الحالي، فإن نافذة مستكشف الحل تعرض ملفات المشروع الحالي والمشاريع الأخرى (شكل 1-6). اختر الأمر Solution Explorer من قائمة View لعرض هذه النافذة.



شكل 1-6: نافذة مستكشف الحل Solution Explorer

في أعلى شجرة ملفات المشروع تلاحظ وجود عنصر المراجع References، يقصد بهذه المراجع المكتبات الخارجية التي تريد تضمينها في المشروع الحالي والوصول إلى فئاتها، انقر بزر الفأرة الأيمن على هذا العنصر واختر الأمر Add Reference من القائمة المنبثقة، لتظهر لك نافذة المراجع (شكل 1-7).



شكل 1-7: نافذة المراجع Reference.

مع العلم انه يمكنك إضافة أو حذف المراجع من هذه النافذة.

نافذة خصائص المشروع Project Property Pages:

عند البدء في إنشاء مشروع جديد، ينصح دائماً بتعديل اعدادات المشروع أولاً عن طريق هذه النافذة، تستطيع الوصول لها بالنقر بزر الفأرة الأيمن على عنصر المشروع في نافذة مستكشف الحل (شكل 1-6 بالصفحة السابقة) ومن ثم اختيار الأمر Properties من القائمة المنبثقة لتظهر هذه النافذة (شكل 1-8 بالصفحة التالية). اكتب اسم المشروع الحالي تحت خانة Assembly Name.



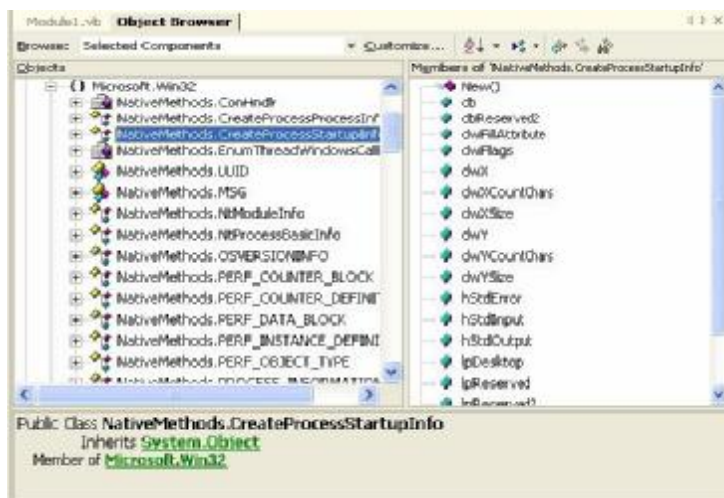
شكل 1-8: نافذة خصائص المشروع.

ملاحظة

في الحقيقة الاسم الرسمي لهذه النافذة هو xxx Property Pages (حيث ترمز الحروف xxx إلى اسم المشروع الحالي)، مع ذلك ستلاحظ أنني استخدم الاسم Project Property Pages في هذا الكتاب، وذلك لجهلي باسم المشروع الحالي عند تطبيقك للأمثلة في جهازك.

نافذة مستعرض الكائنات Object Browser:

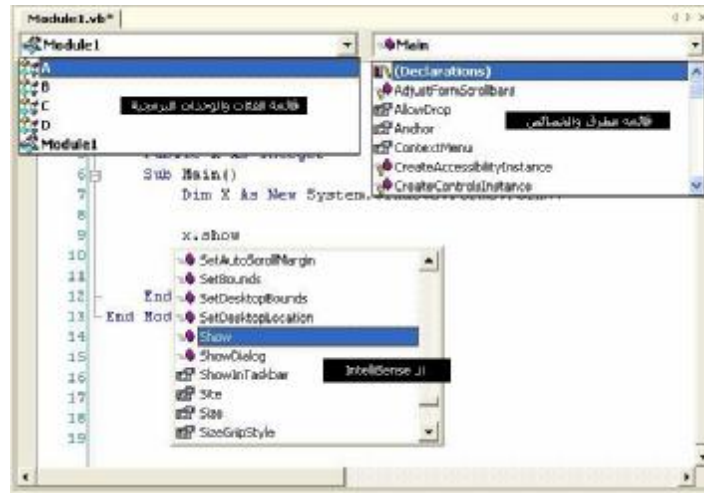
إذا أردت معرفة جميع الفئات والتركيبات والأعضاء التابعة لها سواء كانت هذه الفئات معرفة في المشروع الحالي أو مضمنة من قائمة المراجع Reference، فاختر الأمر View->Other Windows->Object Browser لعرض نافذة مستعرض الكائنات (الشكل 1-9 بالصفحة المقابلة).



شكل 1-9: نافذة مستكشف الكائنات.

نافذة محرر الشيفرة Code Editor:

توفر نافذة محرر الشيفرة (شكل 1-10 بالصفحة التالية) قائمة IntelliSense والتي تظهر بمجرد كتابة النقطة "." بعد اسم الكائن لتعرض جميع أعضائه، كما توجد في أعلى نافذة المحرر قائمتين الأولى لنقل المؤشر إلى الفئات والوحدات البرمجية في الملف الحالي، والثانية للطرق والخصائص. المزيد أيضا، يمكن إخفاء جزء من الشيفرات المصدرية وإظهاره بالضغط على الرموز "+" الموجودة في يسار النافذة.



شكل 1-10: نافذة محرر الشيفرة.

يمكنك تخصيص وتغيير اعدادات هذه النافذة (كالخطوط، الألوان، المحاذاة.... الخ) بالانتقال إلى خانة التبويب Text Editor في صندوق الحوار Option (شكل 1-4 صفحة 19).

القائمة الرئيسية

فيما يلي عرض سريع لمحتويات القائمة الرئيسية بشكل مختصر.

القائمة File:

معظم أوامر هذه النافذة تتعلق بملفات المشروع الحالي، حيث يمكنك من حفظها، فتحها، إغلاقها وإضافة عناصر وملفات أخرى إضافية. كما أنك تستطيع الوصول إلى وظائف الطباعة Printing عن طريق هذه القائمة.

القائمة Edit:

عمليات التحرير كالنسخ Copy، القص Cut، واللصق Paste موجودة في هذه القائمة، بالإضافة إلى مجموعة من أدوات البحث وعلامة الملاحظات Bookmarks.

القائمة View:

تتعلق بإظهار وإخفاء مجموعة كبيرة من النوافذ متعددة الوظائف والأغراض.

القائمة Project:

تتعلق بالمشروع الحالي حيث توفر أوامر لإضافة عناصر وملفات أخرى للمشروع، كما تستطيع الوصول إلى نافذة المراجع Reference أيضا من خلال هذه القائمة. بالنسبة للأمر Set As Startup Project فهو يجعل المشروع الحالي هو المشروع الابتدائي، وذلك في حالة وجود أكثر من مشروع Project في نفس الحل Solution.

القائمة Build:

تمكنك هذه القائمة من ترجمة المشروع Compiling، وبالنسبة للأمر Configuration Manager فهو يحدد الإعدادات المسبقة الحفظ للمترجم.

القائمة Debug:

لحظة تصميم البرنامج تكون عدد عناصر هذه القائمة لا تتجاوز 9 أوامر، ولكن عند التنفيذ سيتم هذا العدد ليصل إلى 13 أمر (بعضها غير مفعّل)، تجد أوامر التنفيذ والإيقاف النهائي والموقت في هذه النافذة، كما تحتوي على جميع وظائف التنقيح Debugging للبرنامج.

انظر أيضا

لعرض بضعة أوامر من هذه القائمة وأدوات التنقيح، انتقل إلى الفصل السابع اكتشاف الأخطاء.

القائمة Tools:

تحتوي على أوامر إضافية مختلفة الوظائف، كما يمكنك جعلها منصة لتشغيل برامج أخرى تستخدمها بشكل متكرر عن طريق اختيار الأمر Externals Tools. وبالنسبة للإضافات Add-Ins، فيمكن الوصول لها عن طريق اختيار الأمر Add-In Manager.

القائمة Window:

لا تعليق!

القائمة Help:

لا بد من أن تكون قد ثبت نسخة من مكتبة MSDN أو .NET Documentation. حتى تتمكن من الوصول إلى التعليمات. وبالنسبة للأمر Dynamic Help، فهو يعرض لك نافذة تظهر تعليمات فورية بمجرد النقر بزر الفأرة على أي عنصر أو نافذة من بيئة التطوير. أنصحك بإغلاق هذه النافذة عند عدم الحاجة إليها، فهي تسبب بطء نسبي في الجهاز خاصة ان كان جهازك بطيء.

أشرطة الأدوات

أشرطة الأدوات Toolbars ما هي إلا أوامر مثل الموجودة في القائمة الرئيسية تقريباً، يمكن إضافتها تحريرها وحذفها بالضغط بزر الفأرة الأيمن على أي شريط واختيار الأمر Customize من القائمة المنبثقة، تماماً كما تفعل مع طاقم تطبيقات Microsoft Office.

كتابة برنامجك الأول

والان سنبدأ بكتابة أول برنامج لك بلغة .NET Visual Basic حتى تتمكن من استخدام بيئة .NET Visual Studio بشكل مبني.

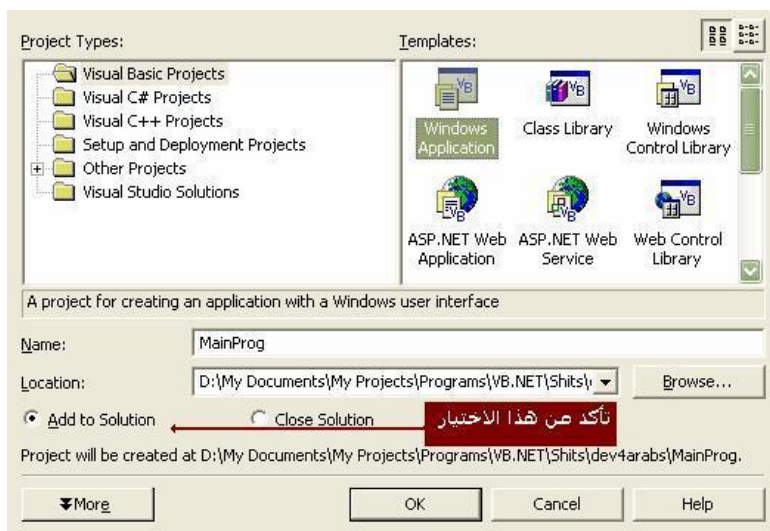
الحلول والمشاريع

الحل Solution هو عبارة عن حاوي لعنصر أو مجموعة عناصر تسمى **المشاريع Projects**، والمشروع هو البرنامج الذي تود إنشائه والذي بدوره يحتوي على عدة عناصر تسمى **ملفات المشروع Project Files** أو وحدات المشروع Project Items.

سأبدأ معك بالحل Solution، يمكنك إنشاء حل جديد باختيار الأمر الفرعي Blank Solution من الأمر New من قائمة File حيث سيظهر لك صندوق حوار بعنوان New Project يطلب منك اسم الحل ومسار مجلده. بعد الضغط على زر OK ستتشئ لك بيئة التطوير .NET Visual Studio -في نفس المجلد الذي حددته سابقاً- بالامتدادين .sln و .suo. الملف .sln هو ملف نصي يمثل مراجع إلى المشاريع المضمنة في الحل الحالي، اما الملف .suo

فيحتوي على الإعدادات وخيارات التخصيص التي تحددها في هذا الحل كمواقع النوافذ المفتوحة وغيرها.

بعد إنشاءك لحل جديد قد تبدأ بإضافة مشروع أو عدة مشاريع إليه، اختر الأمر الفرعي Project من الأمر New من قائمة File سيظهر لك نفس صندوق الحوار السابق، انقر على العنصر Node في الشجرة اليسرى الذي يحمل الاسم Visual Basic Projects حتى تظهر لك في القائمة اليمنى عدة قوالب Templates جاهزة للاستخدام. حدد نوع، اسم، ومسار المشروع ثم تأكد من الاختيار Add to solution قبل الضغط على زر OK (شكل 1-11):



شكل 1-11: القوالب Templates الجاهزة والتي تمثل مشاريع يمكنك البدء فيها.

يمكنك إضافة المزيد من المشاريع بمختلف أنواعها، والطريقة الظرفية التي يمكنك من التحويل بين المشاريع هي نافذة مستكشف الحال Solution Explorer (شكل 1-6 صفحة 27). بالنسبة لملفات المشروع المنجزة بلغة Visual Basic .NET فجميعها تنتهي بالامتداد .vb مهما كان نوعها (سواء ملفات أدوات التحكم UserControl، نماذج Windows Forms، فئات Classes... الخ)، وفي الحقيقة يمكن للملف الواحد أن يحتوي على جميع العناصر السابقة.

أنواع المشاريع

يمكنك Visual Basic .NET من دمج عدة أنواع مختلفة من المشاريع كأدوات التحكم User Controls، تطبيقات قياسية Windows Application، مكتبات الفئات Class Library وغيرها، ربما تكون قد لاحظتها في صندوق الحوار السابق New Projects، واليك يا قارئ العزيز ملخص عنها:

:Windows Application

وهي مشاريع تشابه تطبيقات Windows القياسية (أي Standard Application). الجزء الثالث **تطوير تطبيقات Windows** مخصص لهذا النوع من المشاريع.

:Class Library

هذا النوع من المشاريع يحتوي على مكتبة فئات يمكنك الاستفادة منها في برامج أخرى، كما يمكنك ترجمتها إلى ملفات من النوع DLL.

:Windows Control Library

يمكنك هذا النوع من المشاريع من إنشاء أدوات تحكم User Controls تستخدمها في تطبيقات Windows Application. سنطبق أدوات التحكم بعد مئات الصفحات إلى أن نصل للفصل السادس عشر **مواضيع متقدمة**.

:ASP.NET Web Application

يمكنك من إنشاء مشاريع ASP.NET بحيث تعمل في جهة الخادم Server ويتم عرض صفحاتها عن طريق عملاء Clients بأحد المتصفحات Browsers. الفصلان العشرون والحادي والعشرون **تطبيقات ASP.NET** مخصص لهذا النوع من المشاريع.

:ASP.NET Web Service

هذا النوع من المشاريع يسهل عليك عملية تبادل البيانات عبر الانترنت عن طريق استخدام بروتوكولات HTTP و XML القياسية دون الحاجة إلى تطوير المكونات الموزعة DCOM - كما ستري لاحقا في الفصل الثاني والعشرون **خدمات ويب Web Services**.

:Web Control Library

يمكنك تطوير مشاريع شبيهة بأدوات التحكم لكنها خاصة للعرض على صفحات HTML، وهي مشابهة إلى حد كبير بمشاريع أدوات التحكم User Controls، ولكنها تعرض في المستعرض Browser.

:Console Application

إن كنت تشعر بالحنين إلى تطوير التطبيقات تحت بيئة DOS فهذا النوع مناسب لك تماماً. بالنسبة لنا، سنستمر في تطوير هذا النوع من المشاريع حتى نهاية الجزء الثاني من هذا الكتاب لتعلم أساسيات لغة البرمجة Visual Basic .NET.

:Windows Services

نوع خاص من تطبيقات Windows القياسية بحيث يعمل في الخلفية Background دائماً منذ بداية تحميل نظام التشغيل حتى إغلاق جهاز الكمبيوتر. لي عودة إلى هذا الموضوع في الفصل السادس عشر **مواضيع متقدمة**.

:Empty Project

سهلة جداً ولا تحتاج إلى تفاصيل.

:Empty Web Project

أسهل من سابقتها.

بعد هذه الجولة السريعة حول أنواع المشاريع، يؤسفني إخبارك بأن ما ذكرناه هو مجرد قوالب Templates تقوم بتوليد الشيفرات الضرورية لعمل ما تريد، بل حتى يمكنك إنشاء المزيد من هذه القوالب أو حذف الحالية، فالحل الواحد قد يشمل خدمة Windows Service و برنامج قياسي Windows Application و أداة تحكم User Control، السر كله يكمن في شيفرات ملفات المشروع.

بناء برنامجك الأول

كل الطرق تؤدي إلى روما قيلت سابقا، لديك عشرات الطرق والوسائل التي يمكنك من كتابة برنامجك الأول بـ Visual Basic .NET. ابتداء من ملفات نصية باستخدام المفكرة Notepad أو حتى استخدام احد القوالب الجاهزة -كقالب Console Application. ما يهمني في هذا الفصل إعطائك الخطوات الأساسية لبناء برنامجك الأول باستخدام Visual Basic .NET وليس البدء الفعلي بشرح قواعد ومفردات اللغة، فلن أقدم لك الكثير من التفاصيل، ما انشده هنا هو توضيح عملية بناء المشاريع وتنفيذها فقط. أنشئ أي مشروع جديد ولنقل Console Application على سبيل المثال، ستلاحظ أن نافذة محرر الشيفرة قد فتحت وكتب بها الشيفرة التالية:

```
Module Module1
    Sub Main()

    End Sub
End Module
```

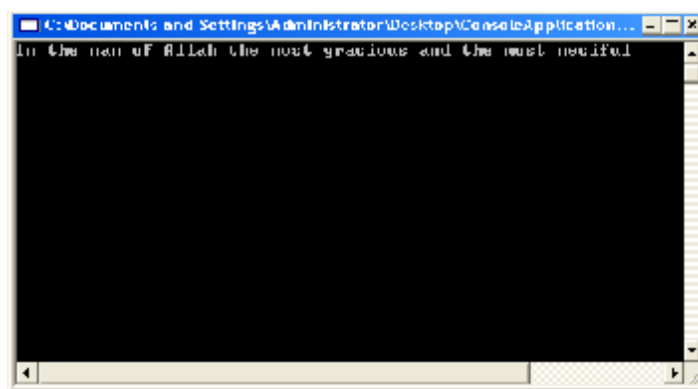
غير اسم الوحدة البرمجية Module1 إلى FirstProg، استخدم الكائن Console لعرض المخرجات على الشاشة. هذا برنامجك الأول وهو يقوم بعرض البسملة لتكون فاتحة خير علينا بمشيئة الله:

```
Module FirstProg
    Sub Main()
        Console.WriteLine("In the name of Allah the most gracious
                           and the most merciful")
    End Sub
End Module
```

اضغط على المفتاح [F5] أو اختر الأمر Start من قائمة Debug لتنفيذ المشروع، (مع العلم أن عملية التنفيذ تؤدي إلى عملية الترجمة Compiling بشكل تلقائي) ستلاحظ أن نافذة سوداء ظهرت واختفت بسرعة، ولكي تتمكن من إيقافها أضف الأمر التالي في البرنامج السابق، بحيث يسمح للمستخدم بالضغط على مفتاح [ENTER] قبل اخفاء النافذة:

```
Module FirstProg
    Sub Main()
        ...
        Console.Read()
    End Sub
End Module
```

اعد تنفيذ البرنامج لترى مخرجاته (شكل 1-12).




شكل 1-12: مخرجات البرنامج الأول.

استخدام ArabicConsole

اضطرت في الفقرة السابقة لاستخدام اللغة الإنجليزية لعرض مخرجات البرنامج الأول وذلك لان الكائن Console لا يدعم الحروف العربية، وبما أنني سأعتمد على هذا الكائن في شرح الشيفرات المصدرية فلا بد من استخدامه، وبدلاً من جعل أمثلة الكتاب تعتمد على الكلمات الإنجليزية، فكرت بتطوير كائن بسيط جداً يحمل الاسم ArabicConsole يحاكي الكائن Console يمكنك من استخدام الحروف العربية. ضع في عين الاعتبار، أن الكائن ArabicConsole لا يحتوي إلا على طريقة واحدة هي WriteLine() فقط، وهي الطريقة الوحيدة التي نحتاجها لعرض المخرجات. ان كنت تود استخدام هذا الكائن في مشاريعك، فيمكنك إضافته عن طريق نافذة المراجع (شكل 1-7) والضغط على الزر Brows ومن ثم البحث عن الملف ArabicConsole.DLL في الدليل الجذري للقرص المدمج.

بعد اضافتك لمرجع الكائن ArabicConsole في مشروعك، تستطيع استخدامه مباشرة وتكتب شيئاً مثل:

```


اضف هذا السطر قبل استخدام الكائن '
' ArabicConsole
Imports ArabicConsoleProject

Module FirstProg
    Sub Main()
        ArabicConsole.WriteLine ("بسم الله الرحمن الرحيم")
    End Sub
End Module

```

انظر أيضا

ستفهم الغرض من استخدام العبارة Imports ArabicConsoleProject في الفصل التالي لغة البرمجة بمشيئة الله.

لست بحاجة لاستخدام الطريقة Read() حتى تمنع النافذة من الإغلاق التلقائي، حيث أن الكائن ArabicConsole يعطيك فرصة لإغلاقها بنفسك.

الترجمة والتوزيع

النقطة الأخيرة التي أريد التطرق لها هي عملية ترجمة البرنامج Compiling وتوزيعه، بالنسبة للترجمة فتوجد عشرات الخيارات المعقدة -بالنسبة لي- تحدد بها سلوك المترجم Compiler، إلا أن فريق التطوير لـ Visual Studio .NET قد سهّلوا علينا عمل ذلك بتجهيز هذه الخيارات بشكل مبدئي بحيث يناسب أغلب الحالات، تعرف هذه الإعدادات بالـ **Configurations**. بشكل مبدئي يمكنك رؤية أحد هذه الإعدادات وهو النص المعنون Debug في شريط الأدوات العلوي لبيئة التطوير Visual Studio .NET (شكل 1-13):



شكل 1-13: إعدادات الترجمة.

بالنسبة للاعدادات Release فحددها إذا كانت هذه آخر مرة تقوم فيها بعملية ترجمة البرنامج وعندما تكون جاهز لتوزيع البرنامج. يمكنك إضافة اعدادات جديدة أو تحرير هذه الاعدادات والتي تجدها في قائمة Configuration Properties من أداة الشجرة الموجودة في صندوق حوار خصائص المشروع Project Property Pages.

أخيراً، إذا أردت توزيع برنامجك إلى أجهزة أخرى عليك إرفاق مكتبة .NET Framework مع البرنامج حتى تعمل، يمكنك إنزال هذه المكتبات من موقع microsoft.com أو يفضل إرفاقها في اسطوانة مستقلة لأن حجمها يزيد عن 100 ميجا بايت.

قد تكون أصبت بالإحباط المبدئي من هذا الفصل ولم تستفد منه الشيء الكثير، ولكن صدقني البداية الحقيقية ستلمسها اعتباراً من الفصل التالي وحتى نهاية الكتاب، وسترى مئات الأسطر من الشيفرات المصدرية مع شرح وافي لكل شيء مبهم لم تفهمه في هذا الفصل، فهدي هنا تقديم جولة سريعة ومبسطة جداً حول تقنية .NET، بيئة التطوير Visual Studio .NET، وطريقة كتابة أول برنامج لك بـ Visual Basic .NET. اقلب الصفحة لتبدأ تعلم لغة البرمجة Visual Basic .NET. من الصفر في الفصل التالي لغة البرمجة.

لغة البرمجة

إن كنت تعتقد بأنني سأبدأ معك في تصميم النوافذ ووضع الأدوات عليها، فيؤسفني إخبارك أن الوقت ما زال مبكراً جداً للحديث عنها، حيث لن أتطرق إلى هذه المواضيع إلا مع بداية الجزء الثالث من هذا الكتاب **تطوير تطبيقات Windows**. ولو كان الأمر بيدي، لنصحت جميع مبرمجي Visual Basic .NET العرب بأن لا يقفزوا إلى برمجة نماذج Windows Forms حتى يتقنوا أساسيات اللغة، فمسألة إتقان لغة البرمجة Visual Basic .NET أمر في غاية الأهمية قبل الانتقال إلى تطوير التطبيقات المختلفة كـ Windows Applications، أو Web Applications، أو Windows Services... الخ، خاصة إن علمت أن Visual Basic .NET ليس موجهاً لتطوير تطبيقات Windows وحسب، بل يمكنك من إنجاز الأنواع المختلفة من المشاريع التي تطرقت إليها في الفصل السابق.

اعتباراً من هذا الفصل وحتى الفصل الثاني عشر سنتعامل مع الكائن ArabicConsole لعرض المخرجات على الشاشة، وتوضيح نتائج الشيفرة لتتعلم لغة البرمجة Visual Basic .NET بشكل سليم. وسأفترض أن لديك خلفية -ولو بسيطة- في البرمجة بغض النظر عن اللغة التي استخدمتها، حيث لن أضيع الكثير من وقتي ووقتكم في شرح المسائل النظرية (كال تكرار، التفرع، المتغيرات، الإجراءات... الخ) فلدي الكثير من المسائل التطبيقية والتي ستوضح لك قضايا أهم بكثير في لغة برمجتك الجديدة Visual Basic .NET.

الوحدات البرمجية Modules

أول مصطلح سنتعرف عليه هو **الوحدة البرمجية Module**، وهي عبارة عن حاوية يمكنك من كتابة جميع شيفراتك المصدرية بداخلها، فلو عدت إلى البرنامج الأول في الفصل السابق، ستلاحظ أننا عرفنا وحدة برمجية باسم FirstProg باستخدام الكلمة المحجوزة Module:

```
Module FirstProg
    Sub Main()
        ArabicConsole.WriteLine("بسم الله الرحمن الرحيم")
    End Sub
End Module
```

لا يمكنك كتابة أي شيفرة خارج نطاق الوحدة البرمجية (أي فوق السطر `Module X` وتحت السطر `End Module`). ولو تجرأت وصرحت عن متغير أو أعلنت عن إجراء خارج نطاق الوحدة البرمجية، فأنت عملياً تكتب خارج نطاق الحاوية، وسيظهر لك المترجم رسالة خطأ

:Statement is not valid in a namespace

```
' لن يتم تنفيذ الشيفرة التالية لعدم وجودها
' داخل وحدة برمجية Module
Dim X As Integer

Sub Test ()
    ...
End Sub

Module FirstProg
    ...
End Module
```

يمكن للملف الواحد أن يحتوي على أكثر من وحدة برمجية، فقد ترغب مثلاً - في تقسيم وحداتك البرمجية استناداً إلى تصنيف وظائفها:

```
Module DrawingFunctions
    ...
End Module

Module InternetFunctions
    ...
End Module

Module SystemFunctions
    ...
End Module

...
...
```

بالنسبة لشروط تسمية الوحدات البرمجية فهي مثل شروط تسمية باقي المعرفات Identifiers الأخرى (كالمتغيرات، التركيبات، الفئات... الخ) وهي:

§ أن لا يزيد عدد حروف المعرف عن 16383 حرف -لا اعتقد انك بحاجة إلى كل هذا العدد!

§ أن يبدأ المعرف بحرف أبجدي، مع ذلك يمكنك استخدام الشرطة السفلية "_" كبدية لاسم المعرف ولكن عليك إتباعها بحرف أبجدي كي يتمكن المترجم من تمييزها عن المعامل "_" (الذي يستخدم لتقسيم الأمر إلى أكثر من سطر).

§ لا يمكنك استخدام اسم يمثل كلمة محجوزة Keyword لتعريف معرف جديد. وان كان لابد من ذلك، فاكتب اسم المعرف داخل القوسين [و] (مثال: Dim [Dim] As Integer). يمكنك استخدام القوسين [و] أيضاً لتعريف معرفات أخرى غير الكلمات المحجوزة.

ملاحظة

تسمح لك لغة البرمجة .NET Visual Basic باستخدام الحروف العربية لكتابة أسماء المعرفات، فهي داعمة لجدول الرموز UNICODE، إلا أنني لم ولن استخدمها لا في هذا الكتاب ولا في مشاريعي الخاصة، فلا اعتقد انك تود رؤية شيفرة مشابهة للشيفرة التالية:

```
فئة As New الكائن Dim
As Integer س Dim

If خاصيته.الكائن = 23 Then
    خاصيته.2.الكائن = س
End If

(س, 20) طريقة.كائن_محضون.الكائن
```

أخيراً، لا تحاول تعريف أكثر من وحدة برمجية Module بنفس الاسم في داخل المشروع، حتى وان اختلفت الملفات التابعة لها، فهذا يسبب خطأ تعارض الأسماء:

لن تتم ترجمة الشيفرة التالية لتعارض ' اسم الوحدة MyModule في الملفين '

```
' First.vb في الملف
Module MyModule
    ...
End Module

' Second.vb في الملف
Module MyModule
    ...
End Module
```

ملاحظة

يمكنك استخدام نفس الاسم لتعريف أكثر من وحدة برمجية شريطة تعريفها في مجالات أسماء Namespaces مختلفة. سأتطرق إلى مجالات الأسماء في القسم الأخير لهذا الفصل.

الإجراء Sub Main()

تستطيع تعريف عدد غير محدود من الإجراءات في داخل الوحدة البرمجية Module بما لـ وطاب لك من الأسماء التي تريدها، إلا أن الإجراء الذي يحمل الاسم Main() له طابع خاص، فهو يمكنك من تخصيص المترجم ليقوم باستدعاء هذا الإجراء مع بداية تنفيذ البرنامج:

```
Module Module1
    Sub Main()
        ArabicConsole.WriteLine("بداية البرنامج من هنا")
    End Sub
End Module
```

المزيد أيضاً، تستطيع تعريف أكثر من إجراء Main() في أكثر من وحدة برمجية:

```
Module Module1
    Sub Main()
        ArabicConsole.WriteLine("من الوحدة البرمجية الاولى")
    End Sub
End Module
```

```
Module Module2
    Sub Main()
```

```

        ArabicConsole.WriteLine("من الوحدة البرمجية الثانية")
    End Sub
End Module

```

والسؤال الذي يطرح نفسه، أي من الإجرائين Main() السابقين سيتم استدعائه مع بداية تنفيذ البرنامج؟ والجواب هو كائن الوحدة الذي تحدده في خانة Startup Object من نافذة Project Property Pages (شكل 2-1). ستظهر رسالة خطأ إذا اخترت وحدة برمجية Module لم يعرف بها إجراء باسم Main()، كما ستظهر نفس رسالة الخطأ ان حددت الاختيار Sub Main (الموجود في نفس القائمة) إن وجد الإجراء Main() في أكثر من وحدة برمجية أو لم يتم تعريفه في أي وحدة برمجية.



شكل 2-1: تحديد الإجراء الابتدائي للمشروع.

الإجراء Sub New()

إجراء آخر له طابع خاص يحمل الاسم New، يعرف هذا النوع من الإجراءات **بالمشيد Constructor**، وهو عبارة عن إجراء يتم تنفيذه بمجرد إنشاء نسخة من الكائن التابع له. فلو كانت الوحدة البرمجية التالية هي أول وحدة يتم تنفيذها في البرنامج، فسيتم تنفيذ الإجراء New() قبل Main():

```

Module Module1
    Sub New()

```

```

        ArabicConsole.WriteLine("سيتم تنفيذ المشيد New اولاً")
    End Sub

    Sub Main()
        ArabicConsole.WriteLine("Main سيتم تنفيذ الإجراء الرئيسي Main")
    End Sub
End Module

```

انظر ايضا

تجد المزيد من التفاصيل والتطبيقات حول المشيدات Constructors في الفصل الرابع **الفئات والكائنات**.

عليك معرفة أن المشيدات في الوحدات البرمجية لا يتم تنفيذها إلا إن قمت باستدعاء احد إجراءات الوحدة البرمجية أو الوصول إلى احد متغيراتها، فالمشيد الموجود في الوحدة البرمجية Module2 التالية لن يتم تنفيذه، وذلك لأننا لم نستخدم أي عضو من أعضاء الوحدة البرمجية التابعة له:

```

Module Module1
    Sub New()
        ArabicConsole.WriteLine("سيتم تنفيذ المشيد New اولاً")
    End Sub

    Sub Main()
        ArabicConsole.WriteLine("Main سيتم تنفيذ الإجراء الرئيسي Main")
    End Sub
End Module

Module Module2
    Sub New()
        ArabicConsole.WriteLine("لن يتم تنفيذ هذا المشيد")
    End Sub
End Module

```

المتغيرات والثوابت

لا يختلف مبرمجان اثنان على أهمية موضوع المتغيرات في أي لغة برمجة، وإذا كان أساس إتقان اللغات الطبيعية هو تعلم حروف ومفردات تلك اللغة، فإن أساس إتقان لغات البرمجة هو تعلم المتغيرات والثوابت التي تبني بها إجراءات برامجك. نظرياً، لا تختلف فكرة المتغيرات في Visual Basic .NET عن لغات البرمجة القديمة، ولكنها تختلف اختلافاً جذرياً في بنيتها التحتية عما كانت عليه في السابق كما سترى لاحقاً.

التصريح عن المتغيرات

ما زالت الكلمة المحجوزة Dim تستخدم للتصريح عن متغير جديد برفقة المعامل As الذي يحدد نوع المتغير:

```
' متغير من النوع Integer
Dim Age As Integer
```

```
' متغيران من النوع String
Dim FirstName As String
Dim LastName As String
```

في الشيفرة السابقة عرفت متغيرين من النوع String في سطرين منفصلين، مع ذلك يمكنك Visual Basic .NET من دمجها في سطر واحد:

```
Dim Age As Integer
Dim FirstName, LastName As String
```

أو دمج تصاريح المتغيرات الثلاثة كلها في سطر واحد -بالرغم من اختلاف أنواعها:

```
' لا انصحك بتعريف انواع مختلفة من
' المتغيرات في سطر واحد
Dim FirstName, LastName As String, Age As Integer
```

ينصح دائماً بتحديد نوع المتغير عند التصريح عنه، وإن لم تحدد نوع المتغير فسيكون نوعه بشكل مبدئي Object، وسيتم تحويله إلى نوع آخر تماثل نوع القيمة التي تسند لها إليه:

```
Dim X

X = 10      ' Integer اصبح المتغير هنا من النوع
X = "10"    ' String وهنا اصبح
```

مع أن الطريقة السابقة تعطيك مرونة كبيرة في تشكيل وتغيير نوع المتغير من وقت لآخر، إلا أنها تسبب بطناً كبيراً في عملية تنفيذ البرنامج، والسبب هو اضطرار المترجم إلى القيام بجهد إضافي لتحويل نوع المتغير.

العبارة Option Explicit:

مبدئياً، عملية التصريح عن المتغيرات أمر إلزامي عليك قبل استخدام المتغير، أما إن كانت العبارة Option Explicit Off مسطورة في أعلى الملف، فيمكنك استخدام المتغيرات والتعامل معها مباشرة دون الحاجة للتصريح عنها بـ Dim:

```
Option Explicit Off

Module Module1
    Sub Main()
        ' متغير جديد استخدمته مباشرة
        ' دون تعريفه بـ Dim
        programmerName = "تركي العسيري"
        ArabicConsole.WriteLine ( programmerName )
    End Sub
End Module
```

صحيح أن الشيفرة السابقة ستوفر عليك عناء التصريح عن المتغيرات، إلا أن هذا الأسلوب غير محبذ بشكل كبير لدى المبرمجين الجادين، الأخطاء الإملائية هي احد الأسباب:

```
' مخرجات الامر التالي لا شئ بسبب الخطأ الاملائي في
' كتابة اسم المتغير السابق
ArabicConsole.WriteLine ( programerName )
```

سبب آخر قد يجعلك ترفض استخدام العبارة Option Explicit Off وهو أن جميع المتغيرات ستكون بشكل ابتدائي من النوع Object وفي كل مرة تسند قيمة جديدة سيتم تحويل نوع المتغير إلى النوع المماثل للقيمة المسندة إليه، مما يسبب بطناً في عملية التنفيذ. عليك الأخذ بعين الاعتبار أن تأثير العبارة Option Explicit Off يشمل الملف الحالي الذي سطرته فيه العبارة فقط. وبدلاً من كتابتها في كافة ملفات المشروع الأخرى، يمكنك اختيار القيمة Off من قائمة Option Explicit في خانة التثبيت Build من نافذة خصائص المشروع Project Property Pages (شكل 2-2 بالصفاة المقابلة).



شكل 2-2: تغيير قيمة Option Explicit من On إلى Off.

قابلية الرؤية وعمر المتغيرات

قابلية الرؤية **Visibility** أو المدى **Scope** للمتغير تمثل قدرة شيفرة البرنامج على الوصول إلى المتغير واستخدامه، فالمتغير **X** الموجود في الإجراء **MySub1()** التالي، لا يمكنك الوصول إليه واستخدامه من خارج الإجراء:

```
Sub MySub1 ()
    Dim X As Integer

    X = 20
End Sub

Sub MySub2 ()
    ' لا يمثل المتغير X السابق
    ArabicConsole.WriteLine (X)
End Sub
```

أما عمر **Lifetime** المتغير، فتتمثل الفترة التي يظل فيها المتغير محتفظاً بقيمته، فالمتغير **X** الموجود في الشيفرة السابقة، سينتهي ويفقد القيمة **20** التي كان محتفظاً بها بمجرد الانتهاء من تنفيذ الإجراء **MySub1()**. وحتى تفهم الأسلوب الذي يتبعه **.NET Visual Basic** لتطبيق مفهومي قابلية الرؤية والعمر للمتغير، عليك معرفة أنواع المتغيرات من منظور الرؤية والعمر:

المتغيرات المحلية الديناميكية:

المتغيرات المحلية الديناميكية Dynamic Local Variables هي متغيرات يتم الإعلان عنها داخل الإجراءات، وعمر المتغير يبدأ من السطر الذي تصرّح فيه عن المتغير وينتهي بعد الانتهاء من تنفيذ الإجراء. أما بالنسبة لقابلية الرؤية فهي محصورة داخل الإجراء الذي صرّحت فيه فقط. تستخدم الكلمة المحجوزة Dim أيضاً للتصريح عن متغير محلي ديناميكي.

تقترح عليك مستندات Microsoft .NET إتباع أسلوب يسمى smallCase في تسمية هذا النوع من المتغيرات، بحيث تكون الكلمة الأولى صغيرة الحروف small والحرف الأول من الكلمات الأخرى كبير Capital. أمثلة:

```
Dim programmerName As String
Dim userID as Integer
Dim employeeSalary As Decimal
```

بعيدا عن موضوع التسمية، يوجد نوع خاص من المتغيرات المحلية الديناميكية يعرف بالـ Block level Variables، وهي متغيرات يتم تعريفها داخل تركيب Block (كحلقة For ... Next، جملة If ... Then، حلقة Do ... Loop وغيرها). مدى هذه المتغيرات يكون محصوراً داخل التركيب الذي أعلنت فيه عن المتغير، وعمرها مثل عمر المتغيرات المحلية الديناميكية السابقة. هذا متغير يحمل الاسم y عرف داخل حلقة For ... Next:

```
Dim counter As Integer

For counter = 1 To 10
    Dim y as integer
    ...
    ...
Next
```

لا تحاول استخدام المتغير المعروف في داخل تركيب خارج هذا التركيب، فمدى هذا النوع من المتغيرات -كما قلت- محصور داخل التركيب فقط:

```
Dim x As Integer

If x = 0 Then
    Dim y As Integer
    ...
    ...
End If

x = y ' رسالة خطأ
```

كما أنك لن تستطيع استخدام اسم متغير ديناميكي محلي لتسمي به متغير داخل تركيب في نفس الإجراء:

```
Dim a As Integer
Do
    Dim a as integer ' رسالة خطأ
    ...
    ...
Loop
```

مع ذلك، يسمح لك Visual Basic .NET باستخدام نفس أسماء المتغيرات العامة أو على مستوى الوحدة أو حتى أسماء متغيرات أخرى معرفة في تركيب آخر:

```
' افترض انه متغير على مستوى الوحدة أو عام
Dim x As Integer
...
...

If x = 0 Then
    Dim x As String ' ممكن جدا
    ...
    ...
End If

Do
    Dim x As Long ' خذ راحتك
    ...
    ...
Loop
```

نقطة أخيرة هامة، عمر هذا النوع من المتغيرات مستمر حتى نهاية الإجراء وليس نهاية التركيب الذي عرفت فيه، فالمتغير x التالي سيحتفظ بقيمته حتى وان خرجت من تركيب الحلقة For counter2 الذي عرف فيها:

```
Dim counter As Integer
Dim counter2 As Integer

For counter = 1 To 3
    For counter2 = 1 To 3
        Dim x As Integer ' سيستمر في الاحتفاظ بقيمته
        x = x + 1
        ArabicConsole.WriteLine(x)
    Next
Next
```

مخرجات الشيفرة السابقة ستكون:

```
1
2
3
4
5
6
7
8
9
```

المتغيرات المحلية الستاتيكية:

المتغيرات المحلية الستاتيكية Static Local Variables هي نفس المتغيرات المحلية الديناميكية، لذلك كل ما قلته في الفقرة السابقة ينطبق هنا دون أي اختلاف، باستثناء أن عمرها الافتراضي ابدى (أي يستمر المتغير الاستاتيكي محتفظاً بقيمته حتى نهاية البرنامج أو موت الكائن التابع له)، كما أنك ستستخدم الكلمة المحجوزة Static عوضاً عن Dim للتصريح عن متغير ستاتيكي:

```
Static staticVariable As Integer
```

لا تحاول استخدام المتغيرات الستاتيكية كثيراً، فهي أبداً من المتغيرات الديناميكية، كما أنها تحجز مساحة في الذاكرة طوال فترة عمل البرنامج دون أن يكون هناك حاجة ماسة إليها. قد تستخدم المتغيرات الستاتيكية مثلاً للاحتفاظ بقيمة عداد أو تنفيذ إجراء مرة واحدة:

```
Sub Counter ()
    Static counter As Integer

    counter = counter + 1
    ...
End Sub

Sub PrintData ()
    Static isPrinting As Boolean

    If isPrinting Then
        Exit Sub
    Else
        isPrinting = True
    End If
    ...
End Sub
```

أخيراً، الكلمة المحجوزة Static لا تطبق إلا على المتغيرات المحلية، فلا تحاول استخدامها مع المتغيرات على مستوى الوحدة أو المتغيرات العامة فهي ستأتيك بطبيعتها.

المتغيرات على مستوى الوحدة البرمجية والمتغيرات العامة:

قد تود من الإجراءات المختلفة التابعة لوحدة برمجية معينة مشاركة المتغيرات فيما بينها، يمكنك Visual Basic .NET من عمل ذلك عن طريق تصريح متغيرات على مستوى الوحدة Module Level Variables، وبهذا يكون مدى هذه المتغيرات شاملاً لجميع إجراءات الوحدة البرمجية. استخدم الكلمة المحجوزة Private أو Dim لتعريف متغير على مستوى الوحدة شريطة أن يتم التصريح عن المتغير خارج الإجراءات:

```
Module Module1
    ' متغيرات على مستوى الوحدة
    Dim x As Integer
    Private y As Integer

    Sub Main()
        x = 50
        ...
    End Sub

    Sub Test ()
        y = 10
        ...
    End Sub
End Module
```

أما المتغيرات العامة Global Variables فمداها يشمل جميع شوارع وأودية وملفات البرنامج، وليست محصورة لوحدة برمجية معينة. استخدم الكلمة المحجوزة Public لتعريف متغير عام:

```
Module Module1
    Public x As Integer ' متغير عام

    Sub Main()
        x = 5
        ...
    End Sub
End Module
```

```

وحدة برمجية اخرى '
Module Module2
    Sub Test()
        x = 1      ' يمكن الوصول إلى المتغير
    ...
End Sub
End Module

```

ملاحظة

يمكنك أيضا استخدام الكلمة المحجوزة Friend لتعريف متغير عام، ولكنها تختلف عن الكلمة المحجوزة Public في قابلية الوصول إلى المتغير من مشروع خارجي. حيث تحصر Friend مدى المتغير على المشروع الحالي فقط.

وكمعيار للتسمية، تقترح عليك مستندات .NET. استخدام أسلوب PascalCase عند تسمية المتغيرات العامة والأسلوب smallCase للمتغيرات على مستوى الوحدة:

```

Public ProgrammerName As String      ' متغيرات عامة
Friend ClientAge As Integer

Dim programmerName As String          ' متغيرات على مستوى الوحدة
Private clientAge As Integer

```

أخيرا، عمر المتغيرات على مستوى الوحدة أو المتغيرات العامة مستمر حتى نهاية البرنامج أو الكائن التابع الذي صرحت فيه.

أنواع البيانات

يبدو أن الوقت قد حان لأخذ جولة تعريفية حول أنواع البيانات المختلفة التي تستطيع استخدامها في برامجك، ولكن دعني أوضح لك نظرتي الشخصية حول هذه الفقرة:

أنواع البيانات (كـ String، Integer، Long، Date.... الخ) لا تتبع -تقنيا- للغة البرمجة .NET Visual Basic، فهي عبارة عن فئات Classes وتركيبات Structures عرفت في مكتبة فئات BCL (التابعة لإطار عمل .NET Framework). والتي تحدثت عنها في الفصل السابق)، وبعبارة أخرى: كل شيء تراه في شيفراتك عبارة عن كائن Object، فإن كنت من المبرمجين المخضرمين عليك أن تعلم علم اليقين أن جميع المتغيرات التي تصرح عنها وتستخدمها

في برامجك، ما هي إلا كائنات منشأة من فئات أو تركيبات معرفة مسبقاً. فحتى أبسط أنواع المتغيرات مثل Byte، عبارة عن كائن له طرق وخصائص تابعه له. وبما أننا ما زلنا في بداية تعلم أساسيات لغة البرمجة Visual Basic .NET، فلا اعتقد انه من المناسب -حالياً على الأقل- التحدث عن هذه الأنواع قبل استيعاب الكائنات والفئات (وهو موضوع الفصل الثالث الفئات والكائنات)، لذلك كان قرارى النهائي هو تأجيل تفصيل هذه الأنواع إلى الفصل السادس الفئات الأساسية. وإلى أن نلتقي هناك، يعرض لك الجدول التالي ملخص سريع لأنواع البيانات الأولية Primitive Data Types التي يدعمها Visual Basic .NET:

النوع	الحجم	مجال القيمة
Boolean	2 بايت	True (صح) أو False (خطأ).
Byte	1 بايت	عدد صحيح من 0 إلى 255.
Char	2 بايت	حرف واحد من نوع UNICODE.
Date	8 بايت	وقت من الساعة 0:00:00 إلى الساعة 11:59:59، كما يشمل تاريخ من يوم 1 يناير لعام 0001 إلى 31 ديسمبر لعام 9999.
Decimal	16 بايت	عدد صحيح من 0 إلى $\pm 79,228,162,514,264,337,593,543,950,335$ أو عشري من 0 إلى $\pm 7.9228162514264337593543950335$
Double	8 بايت	عدد عشري من $1.79769313486231570E+308$ إلى $4.94065645841246544E-324$ بالنسبة للأعداد السالبة. ومن $4.94065645841246544E-324$ إلى $1.79769313486231570E+308$ بالنسبة للأعداد الموجبة.
Integer	4 بايت	عدد صحيح من $-2,147,483,648$ إلى $2,147,483,647$.

النوع	الحجم	مجال القيمة
Long	4 بايت	عدد صحيح من - 9,223,372,036,854,775,808 إلى 9,223,372,036,854,775,807.
	8 بايت	
Object	4 بايت	جميع القيم والأنواع يمكن حفظها هنا.
Short	2 بايت	عدد صحيح من - 32,768 إلى 32,767.
Single	4 بايت	عدد عشري من 3.4028235E+38 إلى E-
		1.40129845 بالنسبة للأعداد السالبة. ومن E- 1.40129845 إلى 3.4028235E+38 بالنسبة للأعداد الموجبة.
String	10 + (2 * عدد الحروف) بايت	من 0 إلى 2 مليار حرف من نوع UNICODE.

عندما نتمعن النظر في الجدول السابق، سنلاحظ وجود نوعين من البيانات الحرفية هما Char و String. بالنسبة للنوع الأول فهو يمثل حرف واحد فقط من حروف Unicode، لذلك فالمتغيرات من النوع Char لا يمكن أن تحمل قيمة حرفية تزيد عن حرف واحد، كما يشترط استخدام حرف الذيل "c" حتى تميز القيمة الحرفية من النوع Char عن قيمة سلسلة الحروف من النوع String:

```
Dim A As Char
```

```
A = "c"
```

```
A = "c" تركي رسالة خطأ هنا
```

قد تستغرب مدى الجدوى من الاعتماد على المتغيرات من نوع Char بدلا من المتغيرات من نوع String رغم إمكانياتها المحدودة، السبب ببساطة السرعة في التنفيذ والاقترصاد في استهلاك مصادر النظام. حيث أن المتغيرات من نوع Char هي متغيرات من النوع ذات القيمة Value Type Variables بينما المتغيرات من النوع String هي متغيرات مرجعية Reference Type Variables. الفروق بين المتغيرات المرجعية والمتغيرات ذات القيمة هو موضوع الصفحة المقابلة.

المتغيرات المرجعية والمتغيرات ذات القيمة:

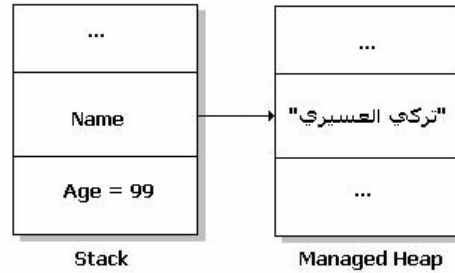
توجد مسائل تقنية بحثة تعتبر في غاية الأهمية تتعلق بالمتغيرات، حيث أتى ذكرت -في بداية هذا القسم من الفصل- أن البنية التحتية للمتغيرات قد تغيرت تغيراً جذرياً عما كانت عليه في لغات البرمجة السابقة، إذ أن القضية ابعء من أن تكون مجرد منطقة محجوزة في الذاكرة تحفظ بها قيمة لتمثل المتغير، فكل شيء تراه في Visual Basic .NET هو كائن Object كما اتفقنا سابقاً. ولكن عليك معرفة أن المتغيرات -أو أنواع البيانات بصفة عامة- في لغة البرمجة Visual Basic .NET تنقسم إلى قسمين رئيسيين مهما اختلفت أنواعها هما: **المتغيرات المرجعية** **Reference Type Variables** و**المتغيرات ذات القيمة** **Value Type Variables**.

سأبدأ معك بالمتغيرات ذات القيمة Value Type Variables، هذا النوع من المتغيرات مشتق وراثياً من الفئة System.ValueType (الوراثة والاشتقاق الوراثي موضوع الفصل الرابع **الوراثة**). تقنياً، هذا النوع من المتغيرات شبيه بفكرة المتغيرات الموجودة في لغات البرمجة السابقة، حيث أن المتغيرات تحفظ قيمها في قسم من ذاكرة البرنامج (قد تكون Stack في معظم الأحوال)، وستمحيى من الذاكرة مباشرة بعد نهاية عمر المتغير. جميع البيانات العددية Numbers، والبيانات من النوع Boolean، Char، و Date، والمتغيرات المعرفة من التركيبات Structures أو Enums هي بيانات من النوع ذات القيمة Value Type.

أما المتغيرات المرجعية Reference Type، فيوجد الكثير لأخبرك به عنها لاحقاً، ولكن كل ما أريد منك أن تعلمه عنها في الوقت الحالي هو أن قيمة المتغير (يسمى **مؤشر Pointer** في هذه الحالة) يتم حفظه كما تحفظ المتغيرات ذات القيمة، بينما تحفظ البيانات الحقيقة للكائن في قسم خاص من ذاكرة البرنامج يسمى **Managed Heap**، وفي الحقيقة لا تتم عملية إزالة قيمها من الذاكرة مباشرة بعد نهاية عمرها الافتراضي، فهي تتطلب عملية تسمى إفراغ المصادر عن طريق المجموعة **Garbage Collection** مقدمة من إطار عمل .NET. فلو كان لدينا هذين المتغيرين:

Dim Name As String = "تركبي العسيري"	متغير مرجعي
Dim Age As Integer = 99	متغير ذات قيمة

يمكننا تخيل مواقعهما بالذاكرة كما في الشكل التالي:



شكل 2-3: أماكن المتغيرات ذات القيمة والمتغيرات المرجعية في الذاكرة.

انظر أيضا

لي عودة أخرى حول المتغيرات ذات القيمة في الفصل السادس
الفئات الأساسية، بينما سيكون لي حديث مطول عن المتغيرات
المرجعية في الفصل الثالث **الفئات والكائنات**.

المتغيرات من النوع String، المصفوفات Arrays، والمتغيرات المعرفة من الفئات Classes جميعها متغيرات مرجعية Reference Type. من منطلق التسلسل التعليمي الذي اتبعه في هذا الكتاب، لا أريد إعطائك جميع الفروقات بين المتغيرات ذات القيمة والمتغيرات المرجعية هنا، حيث أنني أفضل ذكرها في أماكن متفرقة من هذا الكتاب لكي تناسب الفقرات التابعة لها، ولكن دعني أخبرك هنا بأن المتغيرات ذات القيمة أسرع بكثير من المتغيرات المرجعية، كما أنها اقتصادية جدا في استهلاك مصادر النظام، لذلك حاول الاعتماد عليها عوضا عن المتغيرات المرجعية إلا إن دعتك الحاجة لغير ذلك.

إسناد القيم

قد تستغرب من تخصيص فقرة كاملة عن عملية إسناد القيم إلى المتغيرات، إلا أنك ستكتشف أن الأمر بحاجة إلى التطرق لبعض التفاصيل الدقيقة حول هذه المسألة. بادئ ذي بدء، أنت تعلم وأنا أعلم أننا نستطيع إسناد القيم إلى المتغيرات باستخدام معامل إسناد القيم "=", يمكنك إسناد قيمة للمتغير بعد التصريح عنه مباشرة، أو أثناء عملية التصريح بكل انسيابية:

```
Dim X As Integer = 10 ' اسناد قيمة لحظة التصريح
Dim Y As Integer
Dim Z As Long
```

```
Y = 20
Z = 30
```

إن أسندت قيم للمتغيرات أثناء عملية التصريح لأكثر من متغير في سطر واحد، عليك تحديد نوع المتغير لكل تصريح و إلا ستظهر رسالة خطأ:

```
' هنا ممكن
Dim X As Integer, Y As Integer = 20, Z As Long = 30

' رسالة خطأ
Dim X, Y As Integer = 20, Z As Long = 30
```

لست بحاجة لإخبارك انك المسئول الأول والأخير عن مجال القيم التي تسندها إلى المتغيرات، وإن أضفت قيمة خارج نطاق مجال القيم المسموح به لنوع معين من المتغيرات، ستظهر رسالة خطأ وقت التنفيذ:

```
' رسالة خطأ
Dim X As Byte = 256
```

المزيد أيضا، يوفر لك Visual Basic .NET معاملات إضافية لإسناد القيم إلى المتغيرات تعتبر اختصار لعمليات رياضية شائعة توضحها لك هذه الشيفرة:

```
Dim X As Integer = 5 + 5

X += 1 ' x = x + 1
X -= 2 ' x = x - 2
X *= 3 ' x = x * 3
X \= 6 ' x = x \ 6
X ^= 2 ' x = x ^ 2
```

مع العلم أن المعاملات السابقة لا يمكنك استخدامها أثناء عملية التصريح عن المتغير:

```
' لن يسمح لك Visual Basic .NET بكتابة الشيفرة
' بالشكل التالي حتى لو توسط بيل جيتس
Dim X As Integer += 10
Dim Y As Long ^= 20
```

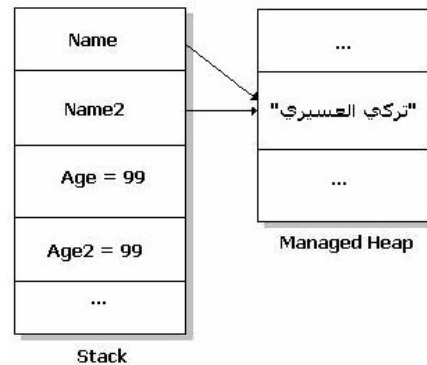
دعني أوضح لك قضية تقنية تتعلق بإسناد القيم حيث تبين لنا أحد الفروق بين المتغيرات ذات القيمة Value Type والمتغيرات المرجعية Reference Type. عملية إسناد القيم بين المتغيرات ذات القيمة تقوم بنسخ فعلي للمتغيرات ليستقل كل متغير بقيمته، أما إسناد القيم بين المتغيرات المرجعية، فهي لا تؤدي إلى نسخ قيم المتغيرات، بل كل ما تقوم به هو نسخ المؤشرات لتشير إلى نفس الكائن الذي يحمل بيانات المتغيرات الفعلية (والموجودة في القسم Managed Heap). إن لم تفهم شيئاً مما سبق، ركز في الشيفرة التالية والموضحة (بالشكل 2-4):

متغيرات مرجعية

```
Dim Name As String = "تركي العسيري"
Dim Name2 As String = Name
```

متغيرات ذات قيمة

```
Dim Age As Integer = 99
Dim Age2 As Integer = Age
```



شكل 2-4: توضيح الفرق في عملية إسناد القيم بين المتغيرات ذات القيمة والمتغيرات المرجعية.

كما ترى في (الشكل 2-4)، رغم أنني عرفت متغيرين مختلفين (Name و Name2) إلا أنهما لا يزالان يشيران إلى نفس القيمة الفعلية للمتغير في القسم Managed Heap، بينما تنقل المتغيرات (Age و Age2) بقيمها في مناطق مختلفة من الذاكرة.

ملاحظة

مؤشرات المتغيرات المرجعية (الموجودة في قسم Stack) حجمها 4 بايت مهما اختلف نوع القيم التي تشير لها.

العبارة Option Strict:

على مر الأجيال السابقة من لغات البرمجة، واجه المبرمجون معاناة كبيرة في مسألة تحويل البيانات المختلفة من القيم. مع ذلك، لن تواجه أي مشاكل عند إتباع أسلوب التحويل الواسع **Widening Conversion**، حيث أن التحويل الواسع يقوم بنسخ قيمة من نوع صغير (Short مثلاً) إلى نوع أكبر منه (كـ Double) مما لا يؤدي إلى التضحية بدقة القيمة:

```
Dim A As Single = 3.9999
Dim B As Double = A
```

```
ArabicConsole.WriteLine(A)      ' 3.9999
ArabicConsole.WriteLine(B)      ' 3.9999
```

وعلى العكس من ذلك، لو كان لدينا متغير من النوع Double وأردت إسناد قيمته إلى متغير آخر من النوع Single، ستفقد هذه العملية الدقة العددية الموجودة في المتغير السابق، السبب واضح لأن المتغيرات من النوع Double تستطيع حمل قيم أكثر دقة من النوع Single:

```
Dim A As Double = 3.99999999
Dim B As Single = A
```

```
ArabicConsole.WriteLine(A)      ' 3.99999999
ArabicConsole.WriteLine(B)      ' 4
```

في الشيفرة السابقة، يضطر Visual Basic .NET إلى إجراء عملية التحويل التلقائية حتى لا يتعدى حدود مجال القيم التي تسمح للمتغيرات من النوع Single حملها، مما يؤدي إلى التضحية بدقة الرقم المحفوظة في النوع Double. يعرف هذا النوع من التحويل **بالضيق Narrowing Conversion**، أي أنك تضيق القيمة من متغير كبير (كـ Double) إلى متغير أصغر منه (وهو Single). أمثلة أخرى: التحويل من Long إلى Integer إلى Short إلى Byte... الخ.

التضييق سبب رئيسي لأمراض البرامج (الشوائب Bugs)، إلا أنك تستطيع استخدام العبارة Option Strict On في أعلى الملف حتى تمنع نفسك كمبرمج و Visual Basic .NET كمتبرمج من إجراء عملية التضييق التلقائية بين الأنواع المختلفة من البيانات، لذلك الشيفرة السابقة ستظهر لك رسالة خطأ إن كانت العبارة Option Strict On مسطورة:

```
' منع عملية التضييق
Option Strict On

Module Module1
    Sub Main()
        Dim A As Double = 1
        Dim B As Single

        A = B      ' التحويل الواسع ممكن

        B = A      ' رسالة خطأ بسبب التضييق

    End Sub
End Module
```

ملاحظة

تأثير العبارة Option Strict يشمل الملف الذي سطرته فيه فقط، وإن أردت شملها في كافة ملفات المشروع الأخرى -دون الحاجة لكتابتها- يمكنك تعديل اعدادات المترجم في نافذة Project Property Pages - كما فعلنا سابقاً مع Option Explicit (شكل 2-2 صفحة 43).

لا يقتصر تأثير العبارة Option Strict على البيانات العددية فقط، بل يمتد ليصل إلى المتغيرات الأخرى كـ Boolean، Date، String... الخ:

```
' ممكن في حالة
' Option Strict Off فقط
Dim A As String
Dim B As Boolean

A = "True"
B = A
```

وعند الحديث عن المعاملات، عليك التفريق بين معامل القسمة \ الخاص بالأعداد الصحيحة، وبين معامل القسمة / والذي يستخدم للأعداد العشرية لان عملية التحويل لن تتم تلقائياً، كما أن معامل الأس ^ يحول القيم إلى Double:

```
' رسالة خطأ في حالة
' Option Strict On
Dim X As Integer
```

```
X = 10 / 2
X = 2 ^ 3
```

في المقابل، تفعيلك للعبارة Option Strict On لا يعني أنك لا تستطيع إسناد الأنواع المختلفة من القيم، بل يمكنك الاستمرار على ذلك شريطة ان تكون لبق وتستخدم دوال التحويل:

```
' ممكن عمل ذلك حتى لو فعلت العبارة
' Option Strict On
```

```
Dim X As Double = 3.2
Dim Y As Integer = CInt(X)
```

يعرض لك الجدول التالي مجموعة من دوال التحويل لأنواع البيانات الأخرى:

الدالة	القيمة التي تعود بها
CBool	Boolean
CByte	Byte
CChar	Char
CDate	Date
CDbl	Double
CDec	Decimal
CInt	Integer
CLng	Long
CObj	Object
CShort	Short
CSng	Single
CStr	String

أخيراً، ما ذكرته في السطور السابقة حول تأثير العبارة `Option Strict On` كان موجهاً إلى المتغيرات ذات القيمة `Value Type` بشكل مباشر، وبالنسبة للكائنات الحقيقية -أقصد المتغيرات المرجعية `Reference Type` - فلها سؤالي وعلوم راجيل أخرى نذكرها لاحقاً في الفصل الثالث **الفئات والكائنات بمشيئة الله**.

الثوابت

بشكل افتراضي، الثوابت العددية الصحيحة يتعامل معها المترجم على أنها من النوع `Integer`، والأعداد العشرية من النوع `Double`:

```
ArabicConole.WriteLine (10) ' Integer قيمة من النوع
ArabicConole.WriteLine (5.5) ' Double قيمة من النوع
```

مع ذلك، يمكنك تحديد نوع الثابت لزيادة سرعة إسناد القيم، فتستطيع استخدام الذيل `"L"` للنوع `Long`، الذيل `"S"` للنوع `Short`، الذيل `"D"` للنوع `Decimal`، والذيل `"F"` للنوع `Single` (راجع بقية الاختصارات في مكتبة MSDN):

```
Dim X As Long
Dim Y As Long

X = 100
Y = 100L ' الإسناد التالي أسرع وذلك لعدم إجراء التحويل الواسع
```

وبالنسبة للثوابت التي تحمل النوع `Date`، فضع قيمة الوقت و/أو التاريخ بين الرمز `#` و `#`:

```
Dim X As Date
Dim Y As Date

x = #1/29/2003#
Y = #2/15/2003 9:30:00 PM#
```

فكرة الثوابت المسماة شبيه بفكرة المتغيرات، إلا أن قيم الثوابت المسماة لا يمكن تعديلها وقت التنفيذ، وذلك لأنها تستبدل بقيمتها أثناء عملية الترجمة للبرنامج، ويتم حفظها في ملف البرنامج النهائي (كـ EXE مثلاً). استخدم الكلمة المحجوزة `Const` لتعريف ثابت جديد:

```
Const PROGRAMMER_NAME = "عباس السريع"
ArabicConsole.WriteLine(PROGRAMMER_NAME) ' عباس السريع
```

تحديد نوع الثابت أمر مفضل لزيادة السرعة، بينما يكون إلزامي إن فعلت العبارة Option Strict On:

```
Const PROGRAMMER_NAME As String = "عباس السريع"
```

عودة إلى الثوابت العديدة، يمكنك كتابة الأعداد بالصيغة الست عشرية Hexadecimal أو الثمانية Octal باستخدام الرموز &H و &O -على التوالي- قبل العدد:

```
' صيغة ست عشرية
ArabicConsole.WriteLine(&HFF) ' 255
```

```
' صيغة ثمانية
ArabicConsole.WriteLine(&O10) ' 8
```

تذكر أن الأعداد -بشكل افتراضي- تكون من النوع Integer، لذلك لا تنسى استخدام الذيل المناسب للقيمة المناسبة، فالحدد التالي يفضل إسناد الذيل "L" له حتى نخرج بالنتيجة المناسبة:

```
ArabicConsole.WriteLine(&HFFFFFFFF) ' -1
ArabicConsole.WriteLine(&HFFFFFFFFL) ' 4294967295
```

أخيراً، لا يمكنك استخدام الصيغة الست عشرية Hexadecimal أو الثمانية Octal للأعداد العشرية:

```
' بودي ولكن لاسف غير ممكن
ArabicConsole.WriteLine(&HFF.25)
```

التركيبات والمصفوفات

عبر الزمن ومع الأيام، ستبدأ بتعريف أنواع خاصة بك في برامجك الجديدة تعرف بالتركيبات، والتي يدعمها Visual Basic .NET بقوة. في هذا القسم من الفصل سأحدثك عن التركيبات من نوع Enums و التركيبات من نوع Structures، كما سأخصص فقرة كاملة حول المصفوفات.

التركيبات من نوع Enums

يمكنك تعريف نوع معين من أنواع المتغيرات بحيث تحصر مجال القيم التي تسندها إليها تعرف بال Enumeration. استخدم الكلمة المحجوزة Enum لتعريف تركيب جديد إما على مستوى

الوحدة البرمجية Module، خارج الوحدة البرمجية، أو داخل تركيب آخر ولكن من النوع Structure. هذا المثال عرفت فيه تركيب يمثل أيام الأسبوع:

```
Enum Day
    Saturday
    Sunday
    Monday
    Tuesday
    Wednesday
    Thursday
    Friday
End Enum
```

والآن يمكنك استخدام التركيب السابق وتعريف متغيرات جديدة منه:

```
Dim x As Day
Dim y As Day

x = Day.Friday
y = x
```

ملاحظة

تقنيا، تصنف التركيبات من النوع Enums ضمن الثوابت، فهي كالثوابت المسماة -التي تطرقت لها سابقا- حيث أن قيمها تستبدل أثناء عملية الترجمة.

عمليا، ستستخدم التركيبات من نوع Enums كوسيطات ترسلها إلى الإجراءات:

```
Sub ShowDay(ByVal CurrentDay As Day)

    If CurrentDay = Day.Friday Then
        ArabicConsole.WriteLine("إجازة")
    End If

    ...

End Sub
```

ثم ترسل إليها المتغيرات من نفس نوع التركيب أو قيم التركيب مباشرة:

```
Dim X As Day
X = Day.Friday
ShowDay(X)
ShowDay(Day.Friday)
```

تبدأ قيم عناصر التركيب من الرقم 0، مع ذلك يمكنك تخصيص قيم أخرى بكل انسيابية:

```
Enum Day
    Saturday = 10
    Sunday = 20
    Monday
    Tuesday
    Wednesday
    Thursday
    Friday
End Enum
```

مع العلم أن مقدار الزيادة لباقي عناصر التركيب هو واحد. أي أن Monday سيحمل القيمة 21، و Tuesday سيحمل القيمة 22، ... وهكذا.

جميع القيم التي عرفت في التركيبات السابقة هي من النوع Integer، مع ذلك يمكنك Visual Basic .NET من تغييرها إلى Byte، Short، أو Long رغم أن مستندات .NET لا تتصحب بعمل ذلك إلا عند وجود سبب مقنع لعمل ذلك:

```
Enum Day As Long
    Saturday
    Sunday
    ...
    ...
End Enum
```

وللحديث عن مدى هذا النوع من المتغيرات، فاختصر عليك الكلام بالقول: ان عرفت التركيب باستخدام الكلمة المحجوزة Private، فان مدى هذا التركيب سيكون محصوراً داخل الوحدة البرمجية الذي عرف فيها هذا التركيب، أما إن استخدمت الكلمة المحجوزة Public أو حتى تجاهلتها فسيكون المدى شاملاً لباقي ملفات المشروع:

```

Module Module1
    ' عام
    Enum GlobalEnum
        Enum1
        Enum2
        ...
    End Enum

    ' عام ايضا
    Public Enum GlobalEnum2
        Enum1
        Enum2
        ...
    End Enum

    ' على مستوى الوحدة البرمجية
    Private Enum PrivateEnum
        Enum1
        Enum2
        ...
    End Enum
    ...
    ...
End Module

```

ملاحظة

يمكنك أيضا تعريف التركيبات من نوع Enum خارج الوحدات البرمجية، إلا أنك لن تستطيع استخدام الكلمة المحجوزة Private في هذه الحالة. أما إن عرفت التركيبات من نوع Enum داخل تركيبات من نوع Structure فالوضع سيكون مثل ما كان عليه مع الوحدات البرمجية Modules.

المزيد أيضا، تستطيع استخدام الكلمة المحجوزة Friend والتي تماثل الكلمة المحجوزة Public، إلا أن الأولى لا تسمح لك استخدام التركيب من خارج حدود المشروع.

التركيبات من نوع Structures

يعرف هذا النوع من التركيبات بالأنواع المعرفة من قبل المستخدم -User Defined Types (UDT)، بحيث يمكنك من دمج أنواع مختلفة من المتغيرات وضمها في تركيب أو كتلة واحدة. استخدم الكلمة المحجوزة Structure لتعريف تركيب جديد:

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

ثم تعرف متغيرات جديدة من هذا التركيب وتتعامل معها كالمتغيرات العادية:

```
Dim Turki As Person

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

ArabicConsole.WriteLine(Turki.Name) ' تركي العسري
ArabicConsole.WriteLine(Turki.Age) ' 99
```

المزيد أيضا، يمكنك نسخ قيم التركيبات بانسيابية كاملة كما تفعل مع المتغيرات العادية، شريطة أن تكون التركيبات متطابقة:

```
Dim Turki2 As Person

Turki2 = Turki

ArabicConsole.WriteLine(Turki2.Name) ' تركي العسري
ArabicConsole.WriteLine(Turki2.Age) ' 99
```

لا تنسى أن التركيبات من نوع Structure يمكن أن تكون متداخلة **Nested** -أي يحتوي بعضها بعضا:

```
Structure Person
    Structure AddressStruct
        Dim City As String
        Dim Countrey As String
    End Structure
    Dim Name As String
    Dim Age As Integer
    Dim Address As AddressStruct
End Structure
```

الوصول إلى عناصر التركيب المحضون يتم من خلال التركيب الحاضن لها بكل منطقية:

```
Dim Turki As Person

Turki.Name = "تركي العسيري"
Turki.Age = 99
Turki.Address.City = "الظهران"
Turki.Address.Countrey = "المملكة العربية السعودية"
```

بالإضافة إلى المتغيرات، عليك معرفة أن التركيبات من نوع Structure في Visual Basic .NET هي تركيبات مطورة ومرنة جدا (مثل التركيبات الموجودة في لغة ++C)، فهي تمكنك من تعريف عناصر إضافية في داخل التركيب كالطرق Methods والخصائص Properties:

```
Structure Person
    Dim Name As String
    Dim Age As Integer

    ' تعريف طريقة
    Sub ShowData()
        ArabicConsole.WriteLine(Name)
        ArabicConsole.WriteLine(Age)
    End Sub
End Structure
```

مرة أخرى، يمكنك الوصول إلى عناصر التركيب واستدعاء طرقه بنفس الطريقة الانسيابية:

```
Dim Turki As Person

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

Turki.ShowData()
```

انظر أيضا

سأتناول الطرق والخصائص بشكل مفصل في الفصل الثالث الغنائ والكائنات.

لا أريد أن أشتت تفكيرك الآن بموضوع الطرق والخصائص (فهو حديث الفصل الثالث كما ذكرت في المربع الأعلى)، ولكن دعني المح لك هنا أن المشيدات Constructors مدعومة بشكل مخفي في التركيبات من النوع Structures. يا الهي! ماذا تقصد يا تركي بكلمة مخفي؟! اقصد يا

عزيزي أن الإجراء Sub New معرف بشكل تلقائي في التركيب دون أن تراه. ولماذا تم إخفاؤه؟ السبب يا سيدي تقني بحث ولا أود أن أبينه إلا في الفصول اللاحقة. حسنا وما الفائدة منه؟ الفائدة ببساطة إسناد قيم ابتدائية لمتغيرات التركيب، فلو حاولت إسناد القيم وقت التصريح كما فعلنا سابقاً عند التصريح عن المتغيرات:

```
Structure MyStruct
    Dim x As Integer = 0
    Dim y As Integer = 10
    ...
End Structure
```

سيظهر لك المترجم رسالة خطأ تفيد بأنك لا تستطيع فعل ذلك (رغم أنني لم أجد سبب منطقي مقنع لا يسمح لي بفعل ذلك)، وهنا يأتي دور المشيد المخفي Sub New() الذي يقوم بإسناد قيم ابتدائية للمتغيرات (0 للمتغيرات العددية، لا شيء للمتغيرات الحرفية، والقيمة Nothing للكائنات). مع ذلك، يمكنك تعريف مشيد Sub New() بنفسك عن طريق تطبيق مبدأ يعرف بإعادة التعريف **Overloading** (وهو حديث الفصل الثالث أيضاً). لعمل ذلك، أضف وسيطات Parameters إضافية مع الإجراء Sub New():

```
Structure Person
    Dim Name As String
    Dim Age As Integer

    ' Overload تمت اعادة تعريفه
    Sub New(ByVal PersonName As String)
        Name = PersonName
        ArabicConsole.WriteLine("تم تنفيذ المشيد")
    End Sub
End Structure
```

رغم أن الوظيفة الأساسية للإجراء Sub New() هي العمل كمشيد، إلا أنه لن يتم استدعائه بمجرد إنشاء كائن من التركيب فيما لو صرحت عن متغير جديد بالطرق التقليدية، والدليل جرب هذا السطر:

```
' مع الاسف الشديد، لن يتم تنفيذ المشيد
Dim Turki As Person
```

ستلاحظ أن Visual Basic .NET لم يقم بتنفيذ ذلك المشيد، والسبب قد يبدو بديهياً إن تذكرت أنه يوجد مشيد Sub New() آخر (لكنه مخفي) تم تنفيذه بدلاً من مشيدنا الظريف. وحتى نبلغ مترجم اللغة أن عليه تنفيذ مشيدنا الجديد، بدلاً من المشيد المخفي علينا استخدام الكلمة المحجوزة New وإرسال الوسيطات التي توافق ذلك المشيد:

```
سيتم تنفيذ المشيد بمجرد التصريح عن المتغير هنا '
Dim Turki As New Person("تركي العسيري")
```

لماذا؟ كيف؟ وما السبب؟ كل هذه الاستفسارات سأنتظر لها في الفصل الثالث الفئات والكائنات بمشيئة الله، لذلك اطلب منك عزيزي القارئ أن لا تقلق نفسك كثيراً بالأشياء الغير مفهومه هنا، حيث أنها تتعلق بالكائنات وطريقة إنشائها، ولا أود نقل مواضيع الفصل الثالث هنا، فنحن ما زلنا نتعلم الأساسيات.

لننتقل إلى موضوع آخر يتمحور حول قابلية الوصول إلى عناصر التركيب، فجميع المتغيرات المحصورة في التركيبات السابقة عرفناها باستخدام الكلمة المحجوزة Dim، لذلك تمكنا من الوصول إلى عناصرها. ولكنك في بعض الأحيان قد تود تعريف متغيرات مخفية لا يمكن الوصول إليها إلا من داخل التركيب نفسه، لذلك عليك استخدام الكلمة المحجوزة Private أثناء التصريح عن متغير في داخل التركيب:

```
Structure Person
    Public Name As String          ' Public مثل Dim هنا
    Dim Age As Integer
    Private MotherName As String

    Sub Test()
        MotherName = "احم احم!"    ' يمكن الوصول إلى المتغير المخفي
        ...                        ' من داخل التركيب فقط
        ...
    End Sub
End Structure
```

الكلمات المحجوزة Public و Private في الشيفرة السابقة، تسمى **محددات الوصول Access Specifiers**. يوجد نوع ثالث من محددات الوصول يستخدم الكلمة المحجوزة Friend وظيفته تماثل وظيفة محدد الوصول Public، إلا أن عناصر التركيب من هذا النوع لا يمكن الوصول إليها من خارج حدود المشروع الحالي (الذي عرف فيه التركيب).

طولناها وهي قصيرة! اختتم فقرة التركيبات من نوع Structure بتوضيح مدى و قابلية الرؤية لها والتي تماثل المدى و قابلية الرؤية للتركيبات من نوع Enums. وحتى أغنيك من عناء قلب الصفحات للبحث عنها، دعني أعيد صياغتها لك هنا: إن عرفت التركيب باستخدام الكلمة المحجوزة Private، فإن قابلية الرؤية لهذا التركيب ستكون محصورة داخل الوحدة البرمجية الذي عرف فيها هذا التركيب، أما إن استخدمت الكلمة المحجوزة Public (أو حتى تجاهلتها) فستكون قابلية الرؤية شاملة لباقي ملفات المشروع. مع العلم انك تستطيع استخدام الكلمة المحجوزة Friend أيضا.

المصفوفات

يمكنك Visual Basic .NET من تعريف المصفوفات سواء كانت أحادية البعد أو متعددة الأبعاد والتي قد تصل إلى 32 بعداً:

```
Dim OneDim (9) As Integer      ' 10 عناصر
Dim TwoDims (1, 1) As String  ' 4 = 2 * 2 عناصر
```

يمكنك فوراً البدء بعملية إسناد القيم لها -كما تفعل مع المتغيرات العادية- مع العلم أن بدء الترقيم لفهرس المصفوفات يبدأ بالرقم 0:

```
OneDim (0) = 100
OneDim (1) = 200
...
OneDim (9) = 900

TwoDims (0, 0) = "تركي"
TwoDims (0, 1) = "العسيري"
TwoDims (1, 0) = "عباس"
TwoDims (1, 1) = "السريع"
```

وان كنت مستعجلاً في عملية إسناد القيم، فإن هذا متاح لك في سطر التصريح مباشرة، شريطة عدم تحديد عدد عناصر المصفوفة:

```
Dim OneDim() As Integer = {1, 2, 3, 4, 5, 6, 7, 8, 9}
Dim TwoDims(,) As String = {{ "تركي", "العسيري" }, { "عباس", "السريع" }}
```

المصفوفات السابقة تسمى مصفوفات ديناميكية Dynamics Arrays لأننا لم نحدد عدد عناصرها، الميزة في هذا النوع من المصفوفات هو إمكانية تغيير حجمها من وقت لآخر باستخدام

الكلمة المحجوزة ReDim، مع الإشارة إلى أن عناصر المصفوفة المعاد تغيير حجمها باستخدام ReDim سوف تلغى:

```
ReDim OneDim (99)
ReDim TwoDims (10, 10)

ArabicConsole.WriteLine ( OneDim(0) ) ' 0
```

مع ذلك، يمكنك تغيير حجم المصفوفة دون المخاطرة بفقد بياناتها باستخدام الكلمة المحجوزة Preserve، ولكن مع الأسف الشديد - لا يمكن تغيير إلا عدد عناصر البعد الأخير فقط في هذه الحالة:

```
' ممكن جدا
ReDim Preserve OneDim (500)
ReDim Preserve TwoDims (10, 500)

' رسالة خطأ
ReDim Preserve TwoDims (500, 500)
```

ومع الأسف الشديد أيضا، لا يمكنك تغيير عدد أبعاد المصفوفة الديناميكية سواء استخدمت Preserve أو لم تستخدمها:

```
' رسالة خطأ
ReDim Preserve OneDim (500, 500)
ReDim TwoDims (100)
```

في المقابل، تستطيع تدمير المصفوفة الديناميكية لتحرير المساحة في الذاكرة في أي وقت تريده باستخدام الأمر Erase:

```
Erase OneDim
Erase TwoDims
```

على صعيد آخر، المصفوفات تعتبر من البيانات المرجعية Reference Type فلا يمكنك نسخ قيمها باستخدام معامل إسناد القيم "=". لي عودة حول المصفوفات في الفصل السادس **الفئات الأساسية**، أما الآن دعني اعرض لك كيف ننسخ قيمة مصفوفة إلى أخرى باستخدام الطريقة Clone():

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

```
' نسخ المصفوفة X إلى Y
Y = X.Clone()
```

انهي حديثي عن المصفوفات بذكر الدالة UBound() التي تعود برقم فهرس العنصر الأخير للمصفوفة، والدالة LBound() برقم الفهرس للعنصر الأول:

```
For counter = LBound(OneDim) To UBound(OneDim)
    ...
Next
```

وبالنسبة للمصفوفات المتعددة الأبعاد، يتوجب عليك إرسال رقم البعد الذي تود معرفة فهرسته:

```
UBound(OneDim)      ' البعد الاول
UBound(OneDim, 1)    ' البعد الاول
UBound(OneDim, 2)    ' البعد الثاني
```

الإجراءات والدوال

يمكنك Visual Basic .NET من تعريف الإجراءات إما بالكلمة المحجوزة Sub أو Function، حيث أن استخدامك للكلمة المحجوزة الثانية يجعل الإجراء قادرا على العودة بقيمة نوعها تحدده عند تعريف الإجراء:

```
' إجراء لا يعود بقيمة
Sub MySub()
    ArabicConsole.WriteLine ("إجراء لا يعود بقيمة")
End Sub
```

```
' دالة تعود بقيمة من النوع Long
Function Abs (ByVal X As Integer) As Long
    If X < 0 Then
        Return -X
    Else
        Return X
    End If
End Function
```

عند استدعاء الإجراءات، عليك كتابة الأقواس حتى لو لم توجد وسيطات Parameters ترسلها لها:

```
' سيقوم احرر باضافة الاقواس ان لم تضيفها '
MySub ()

ArabicConsole.WriteLine (Abs (-5)) ' 5
```

تستطيع إنهاء الإجراء في أي وقت باستخدام الأمر Exit Sub إن تم التعريف باستخدام Sub أو Exit Function إن تم التعريف باستخدام Function:

```
Function Abs (ByVal X As Integer) As Long
    If X = 0 Then
        Exit Function
    End If
    ...
    ...
End Function
```

ملاحظة

إن استخدمت الأمر Exit Function دون تعيين قيمة للدالة، فستعود الدالة بقيمة 0 إن كانت عددية، لا شيء إن كانت حرفية، أو Nothing إن كانت كائنية.

وبالنسبة لقابلية الرؤية فهي إما تكون Private، أو Friend، أو Public كالمتغيرات.

الإرسال بالمرجع أو القيمة

بشكل افتراضي، الوسيطات التي يستقبلها الإجراء هي متغيرات أرسلت بالقيمة، وإن أردت من إجراءك أن تستقبل قيمها بالمرجع لنتمكن من تعديل قيم المتغيرات المرسلة، استخدم الكلمة المحجوزة ByRef:

```
' هنا بالقيمة ولن تتأثر المتغيرات المرسلة '
Sub swapByVal(ByVal a As Integer, ByVal b As Integer)
    Dim temp As Integer
    temp = a
    a = b
    b = temp
End Sub
```

```
' اما هنا بالمرجع وستتأثر المتغيرات المرسله '
Sub swapByRef(ByRef a As Integer, ByRef b As Integer)
    Dim temp As Integer
    temp = a
    a = b
    b = temp
End Sub
```

يتضح الفرق في الشيفرة التالية:

```
Dim A As Integer
Dim B As Integer

A = 10
B = 20

' ارسال بالقيمة
swapByVal (A, B)

ArabicConsole.WriteLine (A) ' 10
ArabicConsole.WriteLine (B) ' 20

' ارسال بالمرجع
swapByRef (A, B)

ArabicConsole.WriteLine (A) ' 20
ArabicConsole.WriteLine (B) ' 10
```

وللحديث عن المسائل التقنية، سأبدأ بعملية إرسال المتغيرات بالقيمة، فهي أبسطاً من الإرسال بالمرجع وذلك لأنه سيتم إنشاء نسخة من البيانات المرسله في كل مرة تستدعي الإجراء. من ناحية أخرى، توجد ميزة في عملية إرسال المتغيرات بالقيمة، وهي عدم التأثير على باقي أجزاء البرنامج أن قمت بتعديل قيمها بطريق الخطأ. أما الإرسال بالمرجع، -كما قلت قبل قليل- هو أسرع من الإرسال بالقيمة، فأنت ترسل مؤشر للمتغير مما يمكنك من تعديل قيمة المتغير المرسل.

ملاحظة

بالنسبة للمتغيرات المرجعية Reference Type Variables المرسله إلى الإجراءات، ستتأثر بالتغييرات حتى وإن أرسلت بالقيمة، أي باستخدام الكلمة المحجوزة ByVal.

تخصيص الوسيطات المرسلة

المزيد من التخصيص حول الوسيطات يوفره لك Visual Basic .NET، إذ يمكنك من التصريح عن الوسيطات الاختيارية Optional والغير محدودة العدد ParamArray.

الوسيطات الاختيارية:

أحيانا نود من إجراءاتك أن تكون مرنة بما فيه الكفاية بحيث لا تشترط توافق عدد المتغيرات المرسلة مع عدد وسيطات الإجراء، تستطيع استخدام الكلمة المحجوزة Optional قبل كل وسيطة اختيارية مع ضرورة تحديد قيمة افتراضية لها في حالة عدم إرسال قيمة للإجراء:

```
Sub MySub(Optional ByVal X As Integer = -1)
    If X = -1 Then
        ArabicConsole.WriteLine ("لم ترسل قيمة")
    End If
    ...
    ...
End Sub
```

نقطة أخرى، لا يمكنك استخدام الكلمة المحجوزة Optional إلا في الوسيطات الأخيرة (أي الموجودة في جهة اليمين) فلا يمكن لوسيطة اختيارية أن تسبق وسيطة عادية:

```
' هكذا ممكن
Sub MySub(ByVal Y As Byte, Optional ByVal X As Integer = -1)
    ...
    ...
End Sub

' انسى هذه الفكرة
Sub MySub(Optional ByVal X As Integer = -1, ByVal Y As Byte)
    ...
    ...
End Sub
```

الوسيطات غير محدودة العدد:

في هذه الحالة فانك لا تحدد عددا معينا للوسيطات التي يستقبلها الإجراء، لان القيم سترسل وتحفظ في مصفوفة تعرفها باستخدام الكلمة المحجوزة ParamArray:

```
Function Sum(ByVal ParamArray Nums() As Integer) As Integer
    Dim counter As Integer

    For counter = 0 To UBound(Nums)
        Sum += Nums(counter)
    Next
End Function
```

تطبيقاً، كل هذه الإستدعاءات صحيحة باستثناء الأخير الذي يتوقع انه اختياري:

```
ArabicConsole.WriteLine ( Sum (1) )           ' 1
ArabicConsole.WriteLine ( Sum (2, 2) )         ' 4
ArabicConsole.WriteLine ( Sum (1, 2, 3, 4, 5) ) ' 15

' خطأ هنا
ArabicConsole.WriteLine ( Sum (1, , 3) )
```

تجاوز الحدود مع Windows API

إن كنت لا تعرف ما هي إجراءات Windows API، فاعتبر نفسك مبرمج محظوظ جداً! وبما انني لست من المبرمجين الشجعان، فلن أتحدث عنها. أما إن كنت من مبرمجي Windows المخصرمين، فتستطيع التصريح عن إجراءات Windows API لتتجاوز حدود عالم إطار عمل .NET Framework.. لعمل ذلك، صرح عن الإجراء باستخدام الكلمة المحجوزة Declare مع تحديد نوع صفحة المحارف إما Ansi، Unicode، أو Auto. إن استخدمت Auto، سيتم تحويل الحروف إلى Unicode تحت جميع الأنظمة باستثناء Windows 98 و Windows ME حيث ستحول إلى Ansi:

```
Module Module1

    Declare Auto Function GetUserName Lib "advapi32.dll" Alias _
        "GetUserNameA" (ByVal lpBuffer As String, _
            ByRef nSize As Integer) As Integer

    Sub Main ()
        ...
        ...
        GetUserName (x, y)
    End Sub

End Module
```

التفرع والتكرار

الخوارزميات من الصعب تطبيقها برمجيا دون استخدامك لجمل التفرع وحلقات التكرار. في هذا القسم من الفصل سنتوغل في عبارات التفرع If ... Then و Select Case، كما سأطرق إلى الحلقات التكرارية المختلفة المتوفرة في لغة البرمجة Visual Basic .NET.

التفرع باستخدام If ... Then

استخدم الكلمة المحجوزة If لتضيف إليها جملة شرطية ثم تلحقها بكلمة Then، ولا تنسى استخدام End If إن وزعت أوامر الشرط في أكثر من سطر (وهو المفضل):

```
' في سطر واحد
If X = 0 Then Y = 1
If X = 1 Then X = 2 : Y = 4
If Y = 1 Then X = 0 Else X = 2

' يفضل توزيعها هكذا
If X = 0 Then
    Y = 1
End If

If X = 1 Then
    X = 0
    Y = 4
End If

If Y = 1 Then
    X = 0
Else
    X = 2
End If
```

ملاحظة

المعامل ":" عكس المعامل "_" بحيث يمكنك من دمج عدة أوامر في سطر واحد.

ذكرت مرتين أن المفضل استخدام الصيغة الموزعة وإغلاقها بـ End If حيث أنها تسهل عليك قراءة وفهم منطق التفرع خاصة إن كانت جمل الشرط متداخلة، ركز معي يا حلو في هذه الجمل:

```
If X = 0 Then
  If Y = 0 Then
    X = 100
  End If
Else
  Y = 1
End If
```

قد يأتي شخص مصمم على اختصار الجمل السابقة في سطر الواحد (فهو مبرمج محترف كما يدعي) ويكتب شيئا مثل:

```
If X = 0 Then If Y = 0 Then X = 100 Else Y = 1
```

أنصحك بعدم الاستماع له مدى الدهر مادام الحمام يغرد! فمنطق التفرع في جملة أخينا في الله خاطئة، حيث أن كلمة Else الأخيرة تتبع للشرط الثاني وليس الأول، أي أن Visual Basic .NET سيفهمها على أنها:

```
If X = 0 Then
  If Y = 0 Then
    X = 100
  Else
    Y = 1
  End If
End If
```

وحتى لا نضيع وقتنا الثمين في مثل هذه السجلات، سأغلق الموضوع بنصيحة: استخدم الصيغة المفردة If ... Then ... End If دائما حتى لو كان جواب الشرط يحتوي على أمر واحد فقط.

أدوات الربط المنطقي:

يمكنك استخدام أدوات الربط المنطقي (And ، Or ، Not ... الخ) بطلاقة كاملة كما تفعل مع لغات البرمجة الأخرى، حيث أنها مدعومة في Visual Basic .NET:

```
If x > 0 And t < 1 Then
  ...
  ...
End If

If Not Y > 1 Then ...
```

دعنا نلهم قليلاً في علم المنطق الرياضي، واطلب منك التركيز في الشرط التالي:

```
If X <> 0 And 10 \ X = 2 Then
```

لغويا، الشرط السابق يختبر قيمة المتغير X ما إذا كانت تساوي الصفر أم لا، وإن كانت لا تساوي صفر فستختبر ناتج القسمة. مع ذلك، فإن الشيفرة السابقة ستظهر رسالة خطأ إن كانت قيمة المتغير X تساوي صفر، والسبب أن Visual Basic .NET سيجري عملية القسمة دائماً. منطقياً، يفترض من Visual Basic .NET أن لا يتعب نفسه ويجري عملية القسمة إن كانت قيمة المتغير X تساوي صفر، والسبب أن الشرط سيكون دائماً False (خاطئ). تقنياً، المعامل And يقوم باختبار جميع الجمل الشرطية التي حوله، لذلك ينصح باستخدام المعامل AndAlso في مثل هذه الحالات:

```
If X <> 0 AndAlso 10 \ X = 2 Then
```

إن كانت قيمة المتغير X في الجملة السابقة تساوي صفر، فإن Visual Basic .NET لن يكمل عملية التحقق من عملية القسمة مما يجنبنا ظهور رسالة الخطأ. إلى جانب المعامل AndAlso يوجد معامل آخر هو OrElse والذي سيتخطى الشرط الثاني إن كان الأول True:

```
' لن يتم التحقق من الشرط الثاني '
' إن كان العدد في المتغير X موجب '
If X > 0 OrElse Y < 0 Then ...
```

عليك معرفة أن المعاملات AndAlso و OrElse تتعامل مع القيم المنطقية فقط، وإن استخدمت الأعداد (ستجرى عملية التحويل التلقائي في حالة Option Strict Off) فستعتبر أي قيمة غير الصفر True، بينما المعاملات And و Or تختبر البتات التي تكون العدد -لذلك تسمى bit-wise operators، فجملة الشرط التالية:

```
x = 3
y = 12
If x <> 0 And y <> 0 Then ... ' True
```

يمكنك اختصارها بالمعامل AndAlso لتعطي نتيجة مماثلة:

```
' عملية المقارنة تختبر القيم
' True And True = True
If x AndAlso y Then ...
```

بينما يؤدي استخدام المعامل And إلى اختبار البتات المكونة للأعداد، لتعطي نتيجة خاطئة:

```
' عملية المقارنة تختبر البتات
' 0011 And 1100 = 0000 (False)
If x And y Then ...
```

أخيراً، يمكنك اختبار مجموعة جمل شرطية وتنفيذ أوامر معينة إن أخفقت كلها باستخدام

:ElseIf

```
If X = 1 Then
...
ElseIf X = 2 Then
...
ElseIf X = 3 Then
...
Else
...
End If
```

التفرع باستخدام Select Case

التفرع باستخدام Select Case يمكن تطبيقه بسهولة تامة:

```
Dim X As Integer
...
...
Select Case X
    Case 1
        ArabicConsole.WriteLine ("محرم")
    Case 2
        ArabicConsole.WriteLine ("صفر")
    ...
    ...
    Case 12
        ArabicConsole.WriteLine ("ذو الحجة")
    Case Else
        ArabicConsole.WriteLine ("غير معرف")
End Select
```

تكمُن قوة العبارة Select Case في تطبيق المعاملات المنطقية أو تحديد مجالات للقيم المطلوب التحقق منها:

```
Dim Grade As Integer
...
...
Select Case Grade
    Case Is < 60
        ArabicConsole.WriteLine ("راسب")
    Case 60 To 69
        ArabicConsole.WriteLine ("مقبول")
    Case 70 To 79
        ArabicConsole.WriteLine ("جيد")
    Case 80 To 89
        ArabicConsole.WriteLine ("جيد جدا")
    Case Is >= 90
        ArabicConsole.WriteLine ("ممتاز")
End Select
```

المزيد أيضاً، يمكنك استخدام الفاصلة بمرونة كاملة، وبنفس المنطق السابق:

```
Dim Letter As Char
...
...
Select Case Letter
    Case "A"c To "Z"c, "a"c To "z"c
        ArabicConsole.WriteLine ("حرف ابجدي")
    Case "0"c To "9"c
        ArabicConsole.WriteLine ("عدد")
    Case ". "c, ":"c, " "c, ";"c, "?"c
        ArabicConsole.WriteLine ("رمز")
    Case Else
        ArabicConsole.WriteLine ("غير معروف")
End Select
```

الفاصلة التي استخدمناها في المثال السابق تمثل أداة الربط Or المنطقية:

```
Select Case True
    Case x > 0, Y < 0
        ' تعادل
        ' If (X > 0) Or (Y < 0)
...
End Select
```

```
Select Case False
    Case x > 0, Y < 0
        ' تعادل
        ' If ( Not (X > 0) ) Or ( Not (Y < 0) )
    ...
End Select
```

الحلقات التكرارية

حدد القيمة الابتدائية والنهائية لحلقة For ... Next

```
Dim counter As Integer
For counter = 2 To 4
    ArabicConsole.WriteLine(counter) ' ثلاث مرات
Next
```

تستطيع التحكم في مقدار الزيادة أو النقصان باستخدام Step:

```
For counter = 5 To 1 Step -1
    ...
Next
```

ضع في اعتبارك أن متغير الحلقة سيزيد أو ينقص بالمقدار المحدد حتى بعد نهاية الحلقة:

```
For counter = 5 To 1 Step -1
    ...
Next
' قيمة العداد 0 وليس 1
ArabicConsole.WriteLine(counter) ' 0
```

المزاح مع متغير الحلقة داخل الحلقة فيه شيء من الخطر، فعدد مرات التكرار للحلقة التالية

هو واحد فقط:

```
For counter = 1 To 100
    counter = 100
    ...
Next
```

وقبل انتهاء المدة الافتراضية للحلقة، يمكنك قطعها وإنهائها بعبارة Exit For:

```

For counter = 1 To 50
    ...
    If y = 10 Then
        Exit For
    End If
    ...
Next

```

حلقة أخرى جميلة جداً تعرف بـ **For Each** تطبق على المصفوفات Arrays أو المجموعات Collections:

```

Dim x(5) As Integer
Dim y As Integer

...

For Each y In x
    ArabicConsole.WriteLine(y)
Next

```

ملاحظة

إن كنت مبتدئاً، فلا تستخدم الحلقة **For Each** كثيراً هذه الأيام حتى تصل إلى الفصل الخامس **الواجهات، التفويض، والمواصفات**، حيث ستجد المزيد من التفاصيل عن استخدام هذه الحلقة.

وعند الحديث عن الحلقات اللانهائية، فلن تجد أفضل من حلقة **Do ... Loop** المرنة جداً، حيث يمكنك من وضع الشرط إما في أعلى الحلقة أو في أسفلها (ليتم تنفيذ أوامر الحلقة مرة واحدة على الأقل في حالة وضع الشرط أسفل الحلقة). إن استخدمت الكلمة المحجوزة **Until**، سيتم تكرار الحلقة حتى يصبح الشرط **True**، أما إن كانت الكلمة المحجوزة **While** هي المرافقة، فسيتم تكرار الحلقة ما دامت قيمة الشرط **True**:

```

Do Until MsgBox("انهي الحلقة؟", MsgBoxStyle.YesNo) = MsgBoxResult.Yes
    ...
Loop

Do While MsgBox("انهي الحلقة؟", MsgBoxStyle.YesNo) = MsgBoxResult.No
    ...
Loop

```

أخيراً، تستطيع في أي لحظة الخروج من الحلقة باستخدام العبارة Exit Do.

التبديل بين For ... Next و Do ... Loop

تستطيع تحويل حلقة For ... Next إلى حلقة Do ... Loop والعكس صحيح، لكن عليك الانتباه إلى أن القيم التي تحددها في بداية الحلقة For ... Next تمثل عدد التكرار حتى وإن تغيرت داخل الحلقة، فبالرغم من أن الحلقتين التاليتين متشابهتين:

```
A = 5
```

```
' حلقة For ... Next
For counter = 1 To A
...
Next

' تحويلها إلى Do ... Loop
counter = 1
Do
...
    counter = counter + 1
Loop Until counter > A
```

إلا أن الاختلاف سيظهر في حال ما إذا تم تغيير قيمة المتغير A، فالحلقة الأولى For ... Next سيتم تنفيذها دائماً خمس مرات حتى وإن تغيرت قيمة المتغير A في داخل الحلقة، بينما تغيير قيمة المتغير يؤثر بشكل كبير على عدد مرات تكرار الحلقة الأخرى Do ... Loop.

مجالات الأسماء Namespaces

الفكرة من مجالات الاسماء Namespaces تقتضي توزيع الأسماء المتشابهة لمعرفات البرنامج (كأسماء الفئات Classes، الوحدات البرمجية Modules، التركيبات Structures ... الخ) إلى كتل مختلفة تسمى مجال أسماء Namespace، بحيث تسهل عليك ترتيب أسمائها على مجموعات، وتمكنك أيضاً من تعريف أسماء متشابهة لمعرفات مختلفة. فلو عرفنا تركيب للفأر باسم Mouse:

```
Structure Mouse
...
...
End Structure
```

فلا يمكننا استخدام نفس الاسم لتعريف تركيب آخر -يمثل جهاز الفأرة- بنفس الاسم Mouse. لذلك سنقوم بتعريف مجالات أسماء مختلفة.

تعريف مجال أسماء

قبل أن تبدأ بتعريف مجالات أسماء خاصة بك، عليك معرفة أن المشروع Project الحالي الذي تصممه قد عرف مجال اسم جديد، ويكون اسمه -بشكل مبدئي- نفس اسم المشروع. يمكنك تغيير مجال الاسم من نافذة Project Property Pages، ومن ثم كتابة اسم مجال الأسماء في خانة Root namespace (شكل 2-5).



شكل 2-5: تسمية مجال الأسماء الجذري للمشروع.

هذا الاسم الذي اخترته يمثل مجال الأسماء الجذري والرئيسي للمشروع الحالي، وجميع المعارف ومجالات الأسماء الأخرى تابعة أو داخلية -إن صح التعبير- ضمن حيز هذا المجال. برمجياً، يمكن تعريف المزيد من مجالات الأسماء في داخل مشروعك باستخدام الكلمة المحجوزة Namespace وتنزيل المجال بالعبارة End Namespace:

```
Namespace Devices
...
...
End Namespace
```

يمكنك البدء بإضافة كل المعارف التي ترغب بحصنها داخل هذا المجال، والمعارف التي يمكنك تعريفها في داخل مجال الأسماء هي إما أن تكون فئات Classes، أو وحدات برمجية Modules، أو تركيبات Structures، أو واجهات Interfaces أو تركيبات من النوع Enumd فقط:

```
Namespace Devices
  Structure Mouse
    ...
    ...
  End Structure

  Structure Printer
    ...
    ...
  End Structure
...
...
End Namespace

Namespace Animals
  Structure Mouse
    ...
    ...
  End Structure

  Structure Cat
    ...
    ...
  End Structure
...
...
End Namespace
```

انظر أيضا

سيتم الحديث عن الفئات Classes في الفصل الثالث **الفئات والكائنات**، والواجهات Interfaces في الفصل الخامس **الواجهات، التفويض، والمواصفات**. أما الوحدات البرمجية Modules والتركيبات - سواء كانت Enums أو Structures فقد فصلتها سابقا في هذا الفصل.

أخيراً، مجالات الأسماء ممكن أن تكون متداخلة Nested:

```
Namespace Devices
  Namespace Inputs
    Structure Mouse
      ...
    ...
  End Structure
  Structure Keyboard
    ...
  End Structure
  ...
End Namespace

Namespace Outputs
  Structure Monitor
    ...
  End Structure
  Structure Printer
    ...
  End Structure
  ...
End Namespace

...
End Namespace
```

الوصول إلى عناصر مجال الأسماء

كل ما هو مطلوب منك تحديد اسم مجال الأسماء ومن ثم ذكر المعرف الذي ترغب في استخدامه:

```
Dim X As Animals.Mouse
Dim Y As Devices.Mouse
...
...
```

وبالنسبة لمجالات الأسماء المتداخلة، عليك ذكر جميع المجالات الحاضنة لها، وبنفس الترتيب المنطقي الذي تتبعه للوصول إلى عناصر تركيبات Structures متداخلة:

```
Dim X As Devices.OutPuts.Printer
Dim Y As Devices.OutPuts.Screen
Dim Z As Devices.Inputs.Keyboard
```

استخدم الاسم الكامل لمجال الأسماء إن كنت خارج المجال فقط (كما في الأمثلة السابقة)، أما إن كنت داخل المجال فلا يوجد داعي لتحديد اسم المجال الحالي:

```
' Devices.Inputs داخل مجال الاسماء
Dim X As Keyboard
Dim Y As Mouse
Dim Z As OutPuts.Printer ' هنا استخدمت مجال خارجي اخر
```

استيراد مجال أسماء باستخدام Imports

يقصد بجمللة استيراد مجال أسماء أي تضمين مجال أسماء معين ودمجه في مجال الأسماء الحالي بحيث يمكنك الوصول إلى جميع عناصره دون الحاجة للالتزام بالصيغة الكاملة لاسم المجال، فالشيفرة التالية:

```
Dim X As Devices.OutPuts.Printer
Dim Y As Devices.OutPuts.Screen
Dim Z As Devices.Inputs.Keyboard
```

يمكنك اختصارها باستيراد المجال Devices باستخدام الكلمة المحجوزة Imports:

```
Imports MyNameSpace.Devices
...
...
Dim X As OutPuts.Printer
Dim Y As OutPuts.Screen
Dim Z As Inputs.Keyboard
```

بل تستطيع اختصارها أكثر من ذلك أيضا باستيراد المجالات الفرعية:

```
Imports MyNameSpace.Devices.OutPuts
Imports MyNameSpace.Devices.Inputs
...
...
Dim X As Printer
Dim Y As Screen
Dim Z As Keyboard
```

ملاحظة

عند استيراد مجال أسماء باستخدام Imports، لابد من كتابة الاسم الكامل لمجال الأسماء (بما في ذلك اسم مجال الأسماء الذي يحتضنه). ففي الشيفرات السابقة، استخدمنا مجال الأسماء الجذري للمشروع MyNameSpace في كل مرة استوردنا فيها مجال أسماء.

قد تفتح شركة استيراد وتصدير في احد الأيام، وتحاول استيراد جميع مجالات الأسماء في برامجك، ولكنك ستصاب بخيبة أمل كبيرة إن حدثت تعارضات، فلو حاولت استيراد هذين المجالين:

```
Imports MyNameSpace.Animals
Imports MyNameSpace.Devices.Inputs

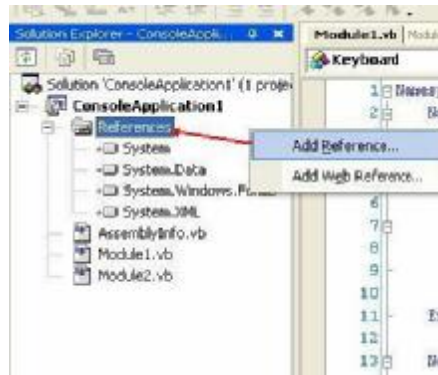
Dim X As Mouse
```

سيظهر لك المترجم رسالة خطأ بسبب تعارض اسم التركيب Mouse في كلا المجالين. وإن كان هذا سبب خسارة لشركة الاستيراد والتصدير الخاصة بك، فيمكنك الالتفاف حول هذا التعارض بتعريف مجال أسماء مؤقت:

```
Imports MyNameSpace.Animals
Imports tmp = MyNameSpace.Devices.Inputs

Dim X As Mouse
Dim Y As tmp.Mouse
```

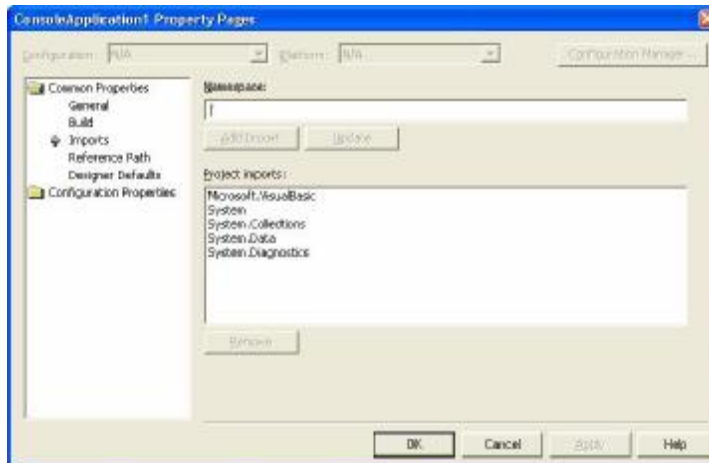
أخيراً، إن أردت استيراد مجالات أسماء لمشاريع وبرامج أخرى (مكتبة فئات .NET Framework) عليك إضافة مرجع لهذه المجالات في خانة المراجع من نافذة مستكشف المشروع Solution Explorer (شكل 2-6 بالصفاة المقابلة).



شكل 2-6: إدراج مراجع إضافية لمجالات أسماء أخرى في المشروع الحالي.

استيراد مجال أسماء دون استخدام Imports

طريقة غير برمجية أخرى يمكنك من استيراد مجالات أسماء (أي دون الحاجة لاستخدام عبارة Imports) وذلك بإضافة جميع مجالات الأسماء التي تود استيرادها في برنامجك عن طريق خانة التوبيخ Imports في صندوق الحوار Project Property Pages (شكل 2-7).



شكل 2-7: استيراد مجال أسماء دون استخدام Imports.

الميزة في هذه الطريقة، هي أن عملية ستشمل كافة ملفات المشروع وليس كاستخدام الكلمة المحجوزة Imports والتي تشمل الملف المسطورة فيه فقط.

كان هذا الفصل نهاية البداية لإتقان لغة البرمجة .NET Visual Basic. تبقى لنا مجموعة من المواضيع الأخرى كالوراثة Inheritance، الواجهات Interfaces، الموصفات Attributes، والتفويض Delegates لنكمل مرحلة تعلم الأساسيات. ولكن قبل ذلك، من المهم جدا استيعاب فكرة الفئات والكائنات عنوان الفصل التالي.

الفئات والكائنات

عند الحديث عن OOP، قد تكون تعاني من عقدة النقص من جملة Visual Basic بين قبائل المبرمجين، أما مع Visual Basic .NET فارفع راسك لفوء أوي أوي أوي! حيث أن Visual Basic .NET لغة برمجة كائنية التوجه OOP حقيقية داعمة لكل سمات لغات OOP الأخرى (باستثناء مبدأ الوراثة المتعددة **Multiple Inheritance** والذي أيضا لا تدعمه أي لغة .NET. أخرى إلا Visual C++ على حد علمي لحظة كتابة هذه السطور).

إن سئمت يوما من الأيام من هذا الكتاب وقررت حرق صفحاته، فدعني اطلب منك عزيز القارئ الاحتفاظ بهذا الفصل على الأقل، حيث انه أهم فصول الكتاب والذي ستحتاجه طيلة حياتك البرمجية مع إطار عمل .NET Framework. وكنصيحة أخوية، لا تنتقل إلى الفصول الأخرى حتى تتقن هذا الفصل إقآن صحيح وتحاول تطبيق كل ما ستتعلمه. مرة أخرى، سأفترض أن لديك خلفية -ولو متواضعة- في البرمجة كائنية التوجه OOP حيث سأبدأ مباشرة في التطبيق دون تقديم مسائل نظرية. حظا سعيدا!

مدخلك السريع للفئات

سيكون هذا القسم من الفصل مدخلك السريع لفهم فكرة الفئات وطرق بنائها. بشكل مبثي، الفئات Classes في Visual Basic .NET مشابهة إلى حد كبير مع التركيبات من النوع Structures، فلو كان لدينا هذا التركيب:

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

يمكنك تحويله إلى فئة باستخدام الكلمة المحجوزة Class عوضا عن Structure:

```

Class Person
    Dim Name As String
    Dim Age As Integer
End Class

```

مع ذلك، فإن نقطة الاختلاف الرئيسية بين الفئات والتركيبات هي أن الفئات من النوع المرجعي Reference Type بينما التركيبات من النوع ذات القيمة Value Type. وبالنسبة لقابلية الوصول لمحتويات الفئة، عليك معرفة أن قابلية الوصول الافتراضية في الفئات هي Private أما مع التركيبات فهي Public، لذلك عليك تحديد محدد الوصول بشكل واضح:

```

Class Person
    ' عدد الوصول Public
    Public Name As String
    Public Age As Integer
End Class

```

والآن يمكنك تعريف متغير (يسمى **مؤشر** إلى كائن في هذه الحالة) من الفئة Person السابقة والبدء بإسناد قيمه، ولكن عليك استخدام الكلمة المحجوزة New عند التصريح عن المؤشر:

```

Dim Turki As New Person()

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

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

```

أما إن أنشأت إجراء (سواء Sub أو Function) داخل هذه الفئة، فمحدد الوصول الافتراضي هو Public كما في التركيبات:

```

Class Person
    Sub PublicMethod () ' Public
    ...
    End Sub

    Public Sub PublicMethod2 () ' Public
    ...
    End Sub

```

```

Friend Sub FriendMethod ()      ' Friend
    ...
End Sub

Private Sub PrivateMethod ()    ' Private
    ...
End Sub
...
End Class

```

على عكس التركيبات، يمكن للفئات أن لا تحتوي على أية أعضاء:

```

' ممكن جدا
Class PersonClass

End Class

' رسالة خطأ
Structure PersonStructure

End Structure

```

ملاحظة

قد تستغرب مدى الجدوى من تعريف فئة لا تحتوي على أية أعضاء، ولكنك قد تفعل ذلك يوما من الأيام إن رغبت بتعريف فئة لاشتقاق واجهات Interfaces من فئات أخرى، سترى لاحقا في الفصل الخامس **الواجهات، التفويض، والمواصفات**.

لا تنسى انه يمكن للفئات أن تكون متداخلة Nested أيضا:

```

Class Car
    Class Engine
        Public Cylinder As Integer
    End Class

    Public Model As String
    Public CarEngine As New Engine()
End Class

```

يمكنك الوصول إلى جميع عناصر الفئة والفئة المحصورة بكل منطقية، شريطة استخدام New عند تعريف متغير من فئة:

```
Dim BMW As New Car()

BMW.Model = "2003"
BMW.CarEngine.Cylinder = 12

ArabicConsole.WriteLine(BMW.Model)           ' 2003
ArabicConsole.WriteLine(BMW.CarEngine.Cylinder) ' 12
```

وللحديث عن قابلية الرؤية Visibility للفئة، فهي افتراضيا Friend إن لم تحدد شيء قبل اسم الفئة، حيث يمكنك الوصول إلى الفئة من داخل المشروع الحالي فقط، بينما Public تجعل الفئة قابلة للاستخدام من مبرمجين آخرين في مشاريعهم الأخرى، أما Private فستحصر قابلية الرؤية على المكان الذي عرفت فيه الفئة مع العلم انك لا تستطيع استخدام الكلمة المحجوزة Private إلا أن عرفت الفئة داخل وحدة برمجية Module أو فئة Class أخرى، أو تركيب من النوع Structure:

```
' Friend
Class FriendClass
...
End Class

Module Module1
' Friend
Class FriendClass2
...
End Class

' Friend
Friend Class FriendClass3
...
End Class

' Public
Public Class PublicClass
...
End Class

' Private
Private Class PrivateClass
...
End Class
...
End Module
```

انظر أيضا

بالإضافة إلى Private، Friend، و Public توجد كلمات محجوزة أخرى تحدد فيها قابلية الرؤية للفئات هي Protected و Protected Friend. مع ذلك، فضلت تأجيل التحدث عنها إلى الفصل الرابع **الوراثة** حيث أنها تتعلق بمبدأ قابلية تطبيق الاشتقاق الوراثي بشكل مباشر. (كل شيء في وقته حلو!)

تستطيع إرسال الفئات على شكل وسيطات إلى الإجراءات، ولكن من الضروري معرفة انك حتى لو أرسلت كائن من فئة بالقيمة (باستخدام ByVal) فالإجراء سيتمكن من تعديل قيمة المتغير المرسل رغم إرساله بالقيمة. وكإثبات لكلامي تحقق من الشيفرة التالية (ستعرف السبب لاحقا في هذا الفصل):

```
Module Module1
    Class TestClass
        Public X As Integer
    End Class

    Sub Main()
        Dim TestObject As New TestClass()

        TestObject.X = 10
        SendByValue(TestObject)
        ArabicConsole.WriteLine(TestObject.X) ' 1 وليس 10
    End Sub
    Sub SendByValue(ByVal obj As TestClass)
        obj.X = -1
    End Sub
End Module
```

أعيد واكرر، الفئات Classes ليست كالتركيبات Structures رغم الشبه الكبير بينهما، وقد ذكرت بضعة فروق بينهما في السطور السابقة لعل أبرزها أن الفئات من النوع المرجعي Reference Type بينما التركيبات من النوع ذات القيمة Value Type، كما أود أن أضيف هنا أن قابلية تطبيق مبدأ الوراثة والاشتقاق الوراثي على الفئات ممكنة، بينما التركيبات لا تصل إلى هذا المستوى الرفيع من البرمجة -كما سترى في الفصل القادم.

أخيرا، تقترح عليك مستندات NET. بعدم إرفاق حرف C عند تسمية الفئات (وهو الأسلوب السائد قديما بين المبرمجين)، حيث أنها تفضل استخدام أسلوب PascalCase لتسمية الفئات

وأعضائها على مستوى Public أو Friend، بينما تستخدم camelCase للأعضاء على مستوى Private:

```
' لا تستخدم طريقة التسمية القديمة '
' CEmployeeData
Class EmployeeData
    Public EmployeeName As String
    Friend EmployeeAge As Integer
    Dim motherName As String          ' Private
    Private salaryAmount As Decimal  ' Private
    ...
    ...
End Class
```

بناء أعضاء الفئات

في القسم السابق عرفتك على الفئات وطريقة بنائها بشكل سريع ومبسط حتى أكون علاقة لطيفة بينك وبينها، أما الآن حان دور التحدث عن كافة التفاصيل المتعلقة ببناء أعضاء الفئات Class Members وهي: الحقول، الطرق، الخصائص، والأحداث.

الحقول Fields

أبسط أنواع الأعضاء التي يمكنك تعريفها في الفئات هي **الحقول Fields**، والحقول -في عالم فئات .NET- ما هي إلا متغيرات تقليدية تعرفها في الفئات مهما اختلف نوعها. لغوياً، الفئة التالية تحتوي على 5 حقول:

```
Class SimpleClass
    Public Field1 As String
    Friend Field2 As Integer
    Dim field3 As Double
    Dim field4 As Boolean
    Private field5 As PersonClass
End Class
```

تتميز الحقول المعرفة في الفئات عن الحقول المعرفة في التركيبات Structure بالقدرة على إسناد قيم لها أثناء عملية التصريح:

```
Class SimpleClass
    Public Field1 As String = "تركي العسيري"
    Friend Field2 As Integer = 99
    ...
End Class
```

شيء جميل آخر، يمكنك تعريف حقل من نفس نوع الفئة، وهذا الأسلوب سيفيدك كثيراً لإجراء خوارزميات برمجية شهيرة (كـ LIFO و FIFO):

```
Class SimpleClass
    Public Field1 As SimpleClass
    Public Field2 As Integer = 0
End Class
```

وعند الرغبة في التعامل مع الحقول السابقة، فكن منطقياً قدر الامكان:

```
Dim SimpleObject As New SimpleClass()

SimpleObject.Field1 = New SimpleClass()
SimpleObject.Field1.Field2 = 10

ArabicConsole.WriteLine(SimpleObject.Field2) ' 0
ArabicConsole.WriteLine(SimpleObject.Field1.Field2) ' 10
```

انظر أيضا

LIFO (Last-In-First-Out) و FIFO (First-In-First-Out) ما هي إلا خوارزميات برمجية تتبع لتنظيم سلسلة من البيانات في كتلة واحدة. يمكنك تطبيقها يدوياً بنفسك إن كنت مستوعباً تماماً للمنطق البرمجي لها. مع ذلك، يوفر لك إطار عمل .NET Framework فئتين هما Stack و Queue لإنجاز هذه الخوارزميات مباشرة تجد شرحها في الفصل السادس **الفئات الأساسية**.

يمكن للحقول أن تصرح بين الأقواس (و)، وبعبارة أخرى الحقول قابلة لأن تعرف على شكل مصفوفات ستاتيكية كانت أم ديناميكية دون أي مشاكل تذكر:

```
Class ArrayClass
    Public X() As Integer
    Public Y(9) As Integer
End Class
```

وعند استخدام المصفوفات الديناميكية، فلست بحاجة لتذكيرك باستخدام ReDim قبل إسناد القيم لها:

```
Dim ArrayObject As New ArrayClass()
```

```
' حقل يمثل مصفوفة ستاتيكية
ArrayObject.X(0) = 100
ArrayObject.X(1) = 200
...
```

```
' حقل يمثل مصفوفة ديناميكية
ReDim ArrayObject.Y(99)
ArrayObject.Y(0) = 10
ArrayObject.Y(1) = 20
...
```

أخيراً، يمكنك حماية حقول الفئة من العبث بها بجعلها للقراءة فقط، استخدم الكلمة المحجوزة `ReadOnly` عند التصريح عن الحقل:

```
Class SimpleClass
    Public ReadOnly CreatedDate As Date = Now()
    ...
End Class
```

استخدامك للكلمة المحجوزة `ReadOnly` سيمنعك منعا باتاً من الكتابة على الحقل بإسناد أي قيمة إليه، مع ذلك فإن فرصة قراءة قيمة الحقل لازالت قائمة:

```
Dim SimpleObject As New SimpleClass()

' رسالة خطأ
SimpleObject.CreatedDate = Now()

' ممكن
ArabicConsole.WriteLine(SimpleObject.CreatedDate)
```

الطرق Methods

كما أن الحقول `Fields` ما هي إلا متغيرات تقليدية، فإن **الطرق Methods** أيضاً ما هي إلا إجراءات (`Subs` أو `Functions`) تقليدية. فالقضية ليست سوى مصطلحات برمجية مقدمة من إطار عمل `.NET Framework`. وبما أنني -بكل تأكيد- لن أعيد فقرات الفصل السابق، دعني أحاول البحث عن أي تلميح من هنا وهناك لها علاقة مباشرة من بعيد أو من قريب بالطرق `Methods`.

قد تفيدك فكرة الطرق بإسناد قيم للخصائص والحقول الأكثر استخداماً مع الفئة، حيث توفر عليك كتابة السطور المكررة لتؤدي إلى زيادة السرعة. فمثلاً، يمكنك تعريف هذه الطريقة التي تسند قيم الحقول الأكثر استخداماً مع الفئة:

```
Class PersonRecord
    Public Name As String
    Public Age As Integer
    Public Address As String

    Sub SetValues(ByVal PersonName As String, _
        ByVal PersonAge As Integer, ByVal PersonAddress As String)

        Name = PersonName
        Age = PersonAge
        Address = PersonAddress
    End Sub
End Class
```

فبدلاً من إسناد قيمة كل حقل على حده، استدعي هذه الطريقة في خطوة واحدة:

```
Dim PersonObject As New PersonRecord

' بدلاً من تعيين قيمة كل حقل على حده
PersonObject.Name = "تركي العسيري"
PersonObject.Age = 99
PersonObject.Address = "المملكة العربية السعودية"

' يفضل استدعاء الطريقة بخطوة واحدة
PersonObject.SetValues("تركي العسيري", 99, "المملكة العربية السعودية")
```

فرق أخير بين الطرق المعرفة في الفئات والطرق المعرفة في التركيبات Structures

يتعلق بقدرة تعريف المتغيرات الستاتيكية في داخل الطريقة، فاستخدام الكلمة المحجوزة Static داخل طرق الفئات ممكن، بينما لا تستطيع استخدامها داخل طرق التركيبات:

```
Class TestClass
    ...
    Sub Method()
        Static X As Integer ' ممكن
    End Sub
    ...
End Class

Structure TestStructure
    ...
    Sub Method()
        Static X As Integer ' رسالة خطأ
    End Sub
    ...
End Structure
```

أما الحديث عن مقترحات مستندات .NET. للتسمية، فهي تقترح استخدام الـ PascalCase عند تسمية الطرق، والـ camelCase عند تسمية وسيطاتها:

```
Sub ShutDown (computerName As String)
    ...
End Sub
```

إعادة التعريف Overloading:

من المبادئ الجميلة في لغات OOP هو مبدأ إعادة التعريف **Overloading**، والذي يمكنك من تسمية طرق مختلفة بنفس الاسم، شريطة اختلاف نوع وسيطات الإجراء:

```
Class SimpleClass
    ' إعادة تعريف الإجراء SameName
    ' ثلاث مرات
    Sub SameName()
        ...
    End Sub

    Sub SameName(ByVal X As Integer)
        ...
    End Sub

    Sub SameName(ByVal Y As String)
        ...
    End Sub
End Class
```

عليك الانتباه انه لا يمكنك إعادة تعريف الإجراء إلا عن طريق تغيير نوع الوسيطات المرسله وليس اسمها، فإعادة تعريف الإجراء SameName() التالي سيظهر رسالة خطأ رغم اختلاف أسماء الوسيطات:

```
Sub SameName(ByVal X As String)
    ...
End Sub

' رسالة خطأ
Sub SameName(ByVal Y As String)
    ...
End Sub
```

ليس هذا فقط، فحتى لو حاولت تغيير طريقة إرسال الوسيطة (إما بالمرجع ByRef أو بالقيمة ByVal) فإن ذلك سيتسبب في ظهور رسالة خطأ أيضاً:

```
Sub SameName(ByVal X As String)
...
End Sub

' رسالة خطأ
Sub SameName(ByRef Y As String)
...
End Sub
```

وكتأكيد لكلامي، لا تحاول أيضاً إعادة تعريف الطرق بتغيير محدد الوصول، فذلك لن يفيد أيضاً:

```
Public Sub SameName(ByVal X As String)
...
End Sub

' رسالة خطأ
Friend Sub SameName(ByVal X As String)
...
End Sub
```

حالة أخرى لا يمكنك من تطبيق مبدأ إعادة التعريف، وهي عند اختلاف نوع الوسيطات الاختيارية Optional فقط للطرق. فمثلاً، إعادة تعريف الطرق التالية سيظهر رسالة خطأ:

```
Sub SameName(Optional ByVal X As Integer = 0)
...
End Sub

' رسالة خطأ
Sub SameName(Optional ByVal X As String = "...")
...
End Sub
```

والسبب أن الاختلاف في وسيطات الطرق هي وسيطات اختيارية Optional فقط. لذلك، لا بد من وجود اختلاف في وسيطات غير اختيارية لتتمكن من إعادة تعريفها.

كما رأيت في الأمثلة السابقة، يمكنك إعادة تعريف الطرق بمجرد تعريفها مباشرة، مع ذلك يفضل استخدام الكلمة المحجوزة Overloads حتى تسهل على مترجم اللغة معرفة الإجراء الذي تود استدعائه، مما يزيد من سرعة التنفيذ:

```
Overloads Sub SameName(ByVal X As String)
...
End Sub

Overloads Sub SameName(ByVal X As Integer)
...
End Sub
```

ملاحظة

كتابة الكلمة المحجوزة Overloads - كما رأيت سابقا - أمر اختياري، ولكنها تصبح أمر إلزامي في باقي الإجراءات (التي تم إعادة تعريفها) إن استخدمتها لأول مرة.

على الجانب الآخر، لنلقي الضوء حول عملية استدعاء الطرق المعاد تعريفها، فلو كانت لدينا هاتين الطريقتين:

```
Class TestClass
    Overloads Sub SameName(ByVal X As Integer)
        ArabicConsole.WriteLine("النسخة التي تستقبل قيمة عددية")
    End Sub

    Overloads Sub SameName(ByVal X As String)
        ArabicConsole.WriteLine("النسخة التي تستقبل قيمة حرفية")
    End Sub
End Class
```

سيتم استدعاء النسخة التي توافق نوع الوسيطة المرسلة:

```
Dim TestObject As New TestClass()
Dim A As Integer = 100
Dim B As String = "100"

TestObject.SameName(A) ' النسخة التي تستقبل قيمة عددية
TestObject.SameName(B) ' النسخة التي تستقبل قيمة حرفية
```

ضع في اعتبارك أن نوع القيمة المرسلة لا يتمثل في نوع المتغير المرسل، وإنما في نوع القيمة النهائية. فمثلا لو استخدمت دوال التحويل سيتم استدعاء الطريقة التي توافق نوع القيمة التي تعود بها الدالة بغض النظر عن نوع المتغير المرسل:

```
Dim TestObject As New TestClass()
Dim A As String = "100"

' سيتم استدعاء النسخة الاولى من
' الطريقة وليس الثانية
TestObject.SameName(CInt(A))
```

وفي حالة إرسالك قيمة لا تكافئ وسيطات الطرق المعرفة، فسيتم استخدام التحويل الواسع Widening Conversion للقيمة ومن ثم استدعاء الطريقة الأقرب للقيمة بعد التحويل، أما التضييق Narrowing Conversion فسيظهر رسالة خطأ:

```
Dim A As Byte = 10
Dim B As Char = "A"c
Dim C As Long = 10

TestObject.SameName(A) ' النسخة التي تستقبل قيمة عددية
TestObject.SameName(B) ' النسخة التي تستقبل قيمة حرفية
TestObject.SameName(C) ' (رسالة خطأ)
```

انظر أيضا

لمزيد من التفاصيل حول دوال التحويل، التحويل الواسع، والتضييق، راجع الفصل الثاني لغة البرمجة.

تطبيقاً، يفيدك مبدأ إعادة التعريف في عدم تكرار كتابة الطرق المتشابهة بأسماء مختلفة، كما انه سبب رئيسي في تقليص عدد سطور جمل الشرط (كـ If ... Then أو Select Case). فمثلاً، قد تود تعريف طريقة لفتح سجل من قاعدة بيانات، تحديد الوسيطة يعتمد اعتماداً كلياً على طريقة البحث (إما بالاسم أو رقم المعرف مثلاً)، فلو قمت بتعريف هاتين الطريقتين:

```
Sub OpenByID (ByVal id As Integer)
...
End Sub

Sub OpenByName (ByVal name As String)
...
End Sub
```

فإن ذلك سيكلفك الكثير من السطور البرمجية في كل مرة تود فتح سجل من قاعدة البيانات، حيث يتوجب عليك أولاً اختبار نوع القيمة (إما حرفية String أو عددية Integer) ومن ثم استدعاء

الطريقة المناسبة، أما مع تطبيق مبدأ إعادة التعريف، فيكفي استخدام الاسم Open لتعريف الطريقة، ومن ثم نقوم باستدعائها مباشرة دون الحاجة للتحقق من نوع القيمة بنفسك:

```
Overloads Sub Open (ByVal id As Integer)
...
End Sub

Overloads Sub Open (ByVal name As String)
...
End Sub
```

أخيراً، يمكنك تطبيق مبدأ إعادة التعريف أيضاً على الإجراءات المعرفة في الوحدات البرمجية Modules والتركيبات من النوع Structures.

المشيدات Constructors:

برمجياً، المشيد **Constructor** هو طريقة Method يتم تنفيذها بمجرد إنشاء نسخة كائن جديدة من الفئة، فلو عرفت إجراء باسم Sub New() في أي فئة:

```
Class TestClass
    Sub New()
        ArabicConsole.WriteLine("تم تنفيذ مشيد الفئة")
    End Sub
End Class
```

فان هذا الإجراء هو مشيد الفئة TestClass وسيتم تنفيذه بمجرد إنشاء نسخة كائن جديدة من الفئة باستخدام الكلمة المحجوزة New:

```
' سيتم تنفيذ المشيد بمجرد انشاء نسخة '
' كائن من الفئة كما بالسطر التالي '
Dim TestObject As New TestClass()
```

تذكر أن المشيد لا يتم تنفيذه إلا عند إنشاء نسخة كائن جديدة من الفئة، وان حاولت إسناد قيم بين متغيرات الكائنات فلا تتوقع تنفيذ المشيد (سأفصل أكثر في موضوع إنشاء وموت الكائنات لاحقاً في هذا الفصل):

```
Dim TestObject As New TestClass() ' سيتم تنفيذ المشيد هنا
Dim TestObject2 As TestClass      ' لن ينفذ المشيد هنا
TestObject2 = TestObject          ' ولا حتى هنا
```

المزيد أيضاً، يمكنك تمرير وسيطات Parameters إلى المشيد لحظة إنشاء الكائن. فمثلاً، قد نود إجبار المبرمج (الذي يستخدم فئتك) أن يعين قيم لبعض حقول الفئة:

```
Class PersonClass
    Public FirstName As String
    Public LastName As String

    ' مشيد يعمل وسيطات
    Sub New(ByVal firstN As String, ByVal lastN As String)
        FirstName = firstN
        LastName = lastN
    End Sub
End Class
```

وعند إنشاء نسخة كائن جديدة من الفئة، عليك إرسال كافة الوسيطات المطلوبة والا فسيكون لرسالة الخطأ نصيب من الظهور:

```
' رسالة خطأ هنا
Dim TestObject As New PersonClass()

' لابد من ارسال الوسيطات للمشيد
Dim TestObject As New PersonClass("العسيري", "تركي")
```

كما تلاحظ في السطر الأول من الشيفرة السابقة، ستظهر رسالة خطأ إن لم ترسل وسيطات لحظة إنشاء الكائن، مع ذلك يمكنك جعل هذه المسألة اختيارية لمستخدم الفئة، إما بجعل الوسيطات اختيارية Optional، أو تطبيق مبدأ إعادة التعريف Overloading (وهو الحل الأفضل) على المشيد:

```
' يمكنك جعل وسيطات المشيد اختيارية
Class PersonClass
    ...
    Sub New(Optional ByVal firstN As String = "",
        Optional ByVal lastN As String = "")

        FirstName = firstN
        LastName = lastN
    End Sub
End Class
```

```
' أو تطبيق مبدأ إعادة التعريف '
' وهو المستحسن '
Class PersonClass
    ...
    Sub New()
        ...
    End Sub

    Sub New(ByVal firstN As String, ByVal lastN As String)
        FirstName = firstN
        LastName = lastN
    End Sub
End Class
```

ملاحظة

لا يمكنك استخدام الكلمة المحجوزة Overloads عند إعادة تعريف المشيدات.

على صعيد آخر، استدعاء المشيدات تراجيعيا Recursively غير ممكن، بعبارة أخرى، لا تستطيع استدعاء المشيد من داخل المشيد نفسه لتطبيق الخوارزميات التراجعية **Recursion**:

```
Class TestClass
    Sub New()
        ...
        New() ' رسالة خطأ
        ...
    End Sub
End Class
```

مع ذلك، يسمح لك Visual Basic .NET باستدعاء مشيد آخر تم إعادة تعريفه بإرسال الوسيطات المطابقة له من مشيد آخر مع ضرورة استخدام الكلمة المحجوزة Me (سأتحدث عن كلمة Me لاحقاً في هذا الفصل):

```
Class TestClass
    Sub New()
        Me.New(100) ' ممكن
        ...
    End Sub

    Sub New(ByVal X As Integer)
        ...
    End Sub
End Class
```

وضع في عين الاعتبار، انك لن تستطيع استدعاء المشيد الرئيسي Sub New() من داخل المشيد المعاد تعريفه Sub New(X As Integer)، حيث أن Visual Basic .NET لن يسمح لك بذلك. أخيراً، المشيد هو المكان الوحيد الذي يسمح لك بإسناد قيمة لحقل معرف باستخدام الكلمة المحجوزة ReadOnly:

```
Class TestClass
    Public ReadOnly X As Integer = -1

    Sub New()
        X = 100 ' ممكن جداً
    End Sub
End Class
```

الخصائص Properties

الخصائص **Properties** ما هي إلا حالة تهجين بين الحقول Fields والطرق Methods، والذي اقصدته من كلمة تهجين في هذا السياق هو أن الخصائص تعمل عمل الحقول حيث يمكنك من إسناد وقراءة قيم منها واليها. وفي الوقت نفسه، الخصائص هي إجراءات كالتطرق Methods، حيث تستطيع كتابة الشيفرة بها. استخدم الكلمة المحجوزة Property لتعريف تركيب لخاصية جديدة ولا تنسى تحديد نوع قيمة الخاصية:

```
Class PersonClass
    Property BirthDate() As Date ' خاصية من النوع Date
    ...
End Property
End Class
```

يمكنك إضافة إجرائين في هذه الخاصية، الإجراء الأول هو Get والذي يتم استدعائه عند قيام مستخدم الفئة بقراءة قيمة الخاصية، أما الإجراء Set فسيتم استدعائه عند إسناد قيمة جديدة للخاصية، كما يفضل استخدام متغير وسيط خاص Private يحمل قيمة الخاصية:

```
' متغير وسيط يحمل قيمة الخاصية
Private m_BirthDate As Date

Property BirthDate() As Date
    Get
        Return m_BirthDate
    End Get
```

```

' Value لا بد من ان تطابق نوع قيمة الخاصية مع الوسيطة
' ByVal وهو Date كما يشترك ارسالها بالقيمة
Set(ByVal Value As Date)
    m_BirthDate = Value
End Set
End Property

```

في الجهة المقابلة، تستطيع التعامل مع الخصائص كما تتعامل تماما مع الحقول، حيث يمكنك إسناد القيم لها وقراءتها في أي وقت:

```

Dim Turki As New PersonClass()

Turki.BirthDate = #1/1/1903#
ArabicConsole.WriteLine(Turki.BirthDate)

```

سيناريو تنفيذ الاجرائين Get و Set السابقين سيكون كالتالي: في كل مرة تسند قيمة إلى الخاصية BirthDate سيتم استدعاء الإجراء Set وإرسال القيمة التي أسندت للخاصية إلى وسيطة الإجراء Value (والتي لا بد من ان تتوافق مع نوع الخاصية وان ترسل بالقيمة ByVal)، وعلى العكس، في كل مرة تقوم فيها بقراءة قيمة الخاصية BirthDate سيتم تنفيذ الإجراء Get والذي -في أغلب الأحوال - سيعود بقيمة الخاصية باستخدام الامر Return كما تفعل مع الدوال Functions. وكحماية لقيمة الخاصية من الكتابة، تستطيع جعلها للقراءة فقط لتسمى **Read Only Property**، يتم ذلك باستخدام الكلمة المحجوزة ReadOnly وحذف إجراء الخاصية Set (فلم يعد هناك داعي من وجوده لأنك لن تستطيع إسناد قيمة للخاصية). وكمثال تطبيقي، يمكنك تعريف خاصية للقراءة فقط تمثل عمر الشخص Age لتعود بقيمة استنادا إلى تاريخ ميلاده (والموجود في الخاصية BirthDate):

```

ReadOnly Property Age() As Integer
    Get
        Return DateDiff(DateInterval.Year, BirthDate(), Now)
    End Get
End Property

```

وكحماية أيضا لقيمة الخاصية من القراءة، يمكنك تعريف خاصية للكتابة فقط، بحيث تمنع مستخدم الخاصية من قراءتها باستخدام الكلمة المحجوزة WriteOnly وحذف الإجراء Get (فلست بحاجة إليه لأنك لن تستطيع إسناد قيمة للخاصية)، قد تستخدم هذا النوع من الخصائص لتمنع مستخدم الفئة من سرقة كلمة المرور مثلا:

```
WriteOnly Property Password() As String
    Set(ByVal Value As String)
        m_Password = Value
    End Set
End Property
```

المزيد أيضا، يمكن للخصائص أن تستقبل وسيطات Parameters كما في الطرق، وذلك بذكر جميع الوسيطات في بداية تعريف الخاصية:

```
Private m_Address(2) As String
Property Address(ByVal index As Integer) As String
    Get
        If index >= 0 And index <= UBound(m_Address) Then
            Return m_Address(index)
        End If
    End Get

    Set(ByVal Value As String)
        If index >= 0 And index <= UBound(m_Address) Then
            m_Address(index) = Value
        End If
    End Set
End Property
```

وعند الرغبة في إسناد قيم للخصائص أو قراءتها، أرسل الوسيطات المطلوبة بين الأقواس:

```
Dim Turki As New PersonClass()
Dim counter As Integer

Turki.Address(0) = "شارع الحزن"
Turki.Address(1) = "حي السعادة"
Turki.Address(2) = "ولاية نيويورك - منغوليا"

For counter = 0 To 2
    ArabicConsole.WriteLine(Turki.Address(counter))
Next
```

أخيرا، يمكنك تطبيق مبدأ إعادة التعريف Overloading على الخصائص أيضا، كما يمكنك تعريف الخصائص في وحدات برمجية Modules وتركيبات من نوع Structure، مع العلم أنك لن تستطيع التصريح عن متغيرات محلية سكونية Static في الخصائص المعرفة في تركيبات Structures.

الخصائص الافتراضية Default Properties:

الخصائص الافتراضية هي خصائص يمكنك إسناد القيم لها أو قراءة قيمها باستخدام اسم الكائن التابعة له فقط ودون الحاجة لذكر اسم الخاصية، وكل ما هو مطلوب منك لجعل الخاصية افتراضية استخدام الكلمة المحجوزة Default لجعل الخاصية افتراضية:

```
Default Property Address(ByVal index As Integer) As String
...
...
End Property
```

وحتى تصل إلى الخاصية Address (التي أصبحت افتراضية)، تستطيع الاكتفاء بذكر اسم الكائن التابع لها دون الحاجة لتحديد اسم الخاصية:

```
Turki(0) = "شارع الحزن"
Turki(1) = "حي السعادة"
Turki(2) = "ولاية نيويورك - منغوليا"

For counter = 0 To 2
    ArabicConsole.WriteLine(Turki(counter))
Next
```

لا تعتقد أنني جعلت الخاصية Address هي الافتراضية لأنها كانت آخر خاصية أدرجت في الفئة السابقة، ولكن السبب الحقيقي هو أنها الخاصية الوحيدة التي تحمل وسيطات Parameters، فلو حاولت جعل الخاصية Age أو الخاصية BirthDate أو حتى الخاصية Password هي الافتراضية، سيظهر لك مترجم .NET Visual Basic رسالة خطأ، فالخصائص السابقة لا توجد بها وسيطات Parameters.

معلومة أخيرة حول الخاصية Address السابقة، إن خطر على بالك حالة معينة لحظة تعريف الكائن Turki السابق، فسأقف ضارباً التحية لعقليتك النبيلة والتي تخرج منها مثل هذه التساؤلات! الحالة التي اقصدها هنا هي عند تعريفك للكائن Turki في مصفوفة:

```
Dim Turki(5) As PersonClass
Set Turki(0) = New PersonClass()
```

كيف ستصل إلى الخاصية Address بشكل افتراضي (أي دون الحاجة لكتابتها)؟ حيث أن الأقواس التي تلي الكائن تمثل رقم فهرس الكائن في المصفوفة، لذلك عليك إضافة أقواس إضافية لتمثل وسيطات الخاصية Address الافتراضية:

```
Turki(0)(0) = "شارع الحزن"
Turki(0)(1) = "حي السعادة"
Turki(0)(2) = "ولاية نيويورك - منغوليا"

For counter = 0 To 2
    ArabicConsole.WriteLine(Turki(0)(counter))
Next
```

أيهم انسب الحقل، الطريقة، أم الخاصية؟

بصراحة شديدة وبدون مقدمات، لا أستطيع إعطائك إجابة منصفة لهذا السؤال، حيث أن القضية تعتمد على الجانب الفكري أكثر من الجانب التقني، إذ أن جميع مشاكلك التي تواجهها يمكن إنجازها برمجياً باستخدام الأنواع الثلاث. فلو أردنا إضافة عضو في فئة يمثل عمر الشخص Age، فجميع هذه الحلول ممكنة:

```
Class PersonClass
    ' حقل
    Public Age As Integer

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

    ' طريقة
    Function Age(Optional ByVal newValue As Integer = -1) As Integer
        Static AgeValue As Integer

        If newValue <= 0 Then
            Return AgeValue
        Else
            AgeValue = newValue
        End If
    End Function
End Class
```

كما ذكرت أن القضية يغلب عليها الطابع الفكري أكثر من التقني، فتحديد نوع العضو يعتمد اعتماداً كلياً على الوظيفة الأساسية لهذا العضو، ليكون القرار النهائي للمبرمج الذي يحدد ما هو النوع المناسب، الحقول في العادة تستخدم لإسناد قيم دون الحاجة إلى أي شيفرة اختبار أو تحقق إضافية (كحقل يحمل قيمة تمثل ملاحظة Note)، وبالنسبة للخصائص فهي كالحقول تقريباً إلا أنها تتطلب شيفرات إضافية للتحقق من القيمة المرسله أو تنفيذ مجموعة من الأوامر لتظهر التأثير المباشر الصادر من عملية إسناد القيم لها (كخاصية عدد الضفادع NumOfFrogs في المستنقع حيث لا يمكن لها أن تكون قيمة سالبة)، أما الطرق فيغلب عليها الجانب العملي والذي يؤدي إلى فعل Action للكائن (كالطريقة Smile لرسم ابتسامة على شفتي كائن).

الأحداث Events

جميع الأعضاء التي تطرقت لها (الحقول، الطرق، والخصائص) يتم بنائها من مؤلف الفئة، وهو المسئول الأول والأخير عن الشيفرات المصدرة التي ملأت الفئة، ولكن في اغلب الأحوال يفضل إعطاء فرصة كبيرة لمستخدم الفئة من تخصيص شيفرات معينة وكتابتها بالطريقة التي يريد لها لتجعل الفئة قابلة أكثر لإعادة الاستخدام Reusable. ولكي يتحقق ذلك، عليك إضافة أعضاء تسمى الأحداث Events.

الأحداث في الفئات ما هي إلا مؤشرات مفوضة إلى إجراءات تقليدية يتم استدعائها لحظة وقوع حدث معين يحدده مؤلف الفئة، ولكن نقطة الاختلاف الجوهرية بين الأحداث وباقي الأنواع الأخرى من أعضاء الفئات، هي أن ردة الفعل أو -عبارة تقنية- الشيفرات التابعة لتلك الإجراءات لا ينجزها إلا مستخدم الفئة وليس مؤلفها. فمثلاً، لو أردت تعريف حدث في الفئة السابقة PersonClass يمثل موت الشخص Die، سأعطي كامل الحرية لمستخدم الفئة لكتابة الشيفرات المصدرة التي يريد لها لحظة وقوع الموت (وقوع الموت يسمى -برمجياً- إطلاق الحدث Fire Event في هذا السياق). لتعريف هذا الحدث، استخدم الكلمة المحجوزة Event:

```
Class PersonClass
    Event Die()
    ...
End Class
```

والآن قد أعطيت حرية كبيرة لمستخدم الفئة من تحديد ردة الفعل وكتابة الشيفرات التي يريد لها لحظة انطلاق الحدث. وحتى يتمكن المستخدم من فعل ذلك، عليه استخدام الكلمة المحجوزة

WithEvents عند التصريح عن متغير الكائن، ومن ثم تحديد الإجراء الذي يود تنفيذه لحظة انطلاق الحدث باستخدام الأمر Handles وإلحاقها بالحدث المراد قنصه:

```
Module Module1
    Dim WithEvents Turki As New PersonClass()

    Sub Main()
        ...
    End Sub

    ' قنص الحدث Die
    Sub PersonHasDied() Handles Turki.Die
        ArabicConsole.WriteLine("لقد توفي الشخص")
    End Sub
End Module
```

ملاحظة

لا تستطيع استخدام الكلمة المحجوزة WithEvents على المتغيرات المحلية Local Variables، حيث يشترط أن يكون المتغير على مستوى الوحدة البرمجية Module، أو على مستوى الفئة Class، أو عام Global.

والسؤال الذي يطرح نفسه بكل تأكيد، متى سيموت Turki؟ والجواب المؤكد هو في علم الغيب والذي لا يعلمه إلا الله عز وجل، لذلك دعني أعيد صياغة السؤال حتى نتحور حول الشيفرة السابقة ليكون: متى سيتم إطلاق الحدث Die حتى يتم تنفيذ الإجراء PersonHasDied؟ والجواب هو في أي وقت يحدده مؤلف الفئة عن طريق الأمر RaiseEvent، فلو عدنا إلى مرحلة بناء الفئة يمكننا أن نعرف الطريقة KillHim() التي تؤدي إلى إطلاق الحدث Die:

```
Class PersonClass
    Event Die()

    Sub KillHim()
        RaiseEvent Die()
    End Sub
    ...
End Class
```

ما أن يقوم مستخدم الفئة باستدعاء الطريقة KillHim() فسيتم إطلاق الحدث Die، وحتى نشاهد هذه اللحظة الحزينة، دعنا نكتب هذه الشيفرة التي تستدعي الطريقة KillHim():

```

Module Module1
    Dim WithEvents Turki As New PersonClass()

    Sub Main()

        ' استدعاء الطريقة يؤدي إلى إطلاق الحدث
        Turki.KillHim()

    End Sub

    Sub PersonHasDied() Handles Turki.Die
        ArabicConsole.WriteLine("لقد توفي الشخص")
    End Sub
End Module

```

المزيد أيضاً، الكلمة المحجوزة `Handles` لا تستخدم لقنص حدث واحد فقط، بل يمكنك إضافة أحداث مختلفة من كائنات مختلفة بحيث تؤدي إلى تنفيذ نفس الإجراء عند انطلاق احد أحداث هذه الكائنات:

```

Sub TestEvents() Handles Turki.Die, Ali.Die, Apple.Stink
    ...
End Sub

```

مع ذلك، لن تستطيع استخدام `Handles` لقنص أحداث لكائنات مختلفة في حالة واحدة وهي عندما تكون الوسيطات `Parameters` للحدث مختلفة، فمثلاً لو عرفت حدث في فئة يستقبل وسيطات إضافية:

```

Class TestClass
    Event TestEvent(ByVal x As Integer)
    ...
End Class

```

عليك استخدام `Handles` في إجراء يطابق عدد ونوع الوسيطات التي يتطلبها الحدث:

```

' لا يمكن هنا
Sub TestSub() Handles TestObject.TestEvent
    ...
End Sub

' هنا مناسب
Sub TestSub(ByVal Y As Integer) Handles TestObject.TestEvent
    ...
End Sub

```

ملاحظة

بالنسبة للأحداث التي تحتوي على وسيطات إضافية، فلا بد من إرسال قيم الوسيطات عند إطلاق الحدث باستخدام RaiseEvent، فالحدث السابق يتم إطلاقه هكذا:

```
RaiseEvent TestEvent(10)
```

قصر الأحداث باستخدام AddHandler:

بالإضافة إلى الكلمة المحجوزة WithEvents لقصر الأحداث، يوفر لك Visual Basic .NET طريقة أخرى أكثر مرونة لقصر الأحداث وهو الأمر AddHandler. الشيء الجميل والذي يعجبني جدا في هذا الأمر، هو أنك تحدد الإجراء الذي تود تنفيذه لحظة إطلاق الحدث وقت تنفيذ وليس التصميم. يتطلب الأمر AddHandler اسم الحدث الذي تود قنصه و مؤشر إلى الإجراء الذي تود تنفيذه (للحصول على مؤشر الإجراء استخدم AddressOf):

```
Module Module1
    Dim Turki As New PersonClass()

    Sub Main()
        ' قنص الحدث
        AddHandler Turki.Die, AddressOf PersonHasDied
        Turki.KillHim()
    End Sub

    Sub PersonHasDied()
        ArabicConsole.WriteLine("لقد توفي الشخص")
    End Sub
End Module
```

سيستمر الإجراء PersonHasDied() السابق في انتظار الحدث Die حتى موت الكائن Turki نهائيا (سأحدث بالتفصيل الممل حول موت الكائنات لاحقا)، وان كنت لا ترغب في الانتظار طيلة هذه الفترة، يمكنك إيقاف عملية قنص الحدث في أي وقت باستخدام الأمر RemoveHandler والذي يستدعي بنفس صيغة AddHandler:

```
RemoveHandler Turki.Die, AddressOf PersonHasDied
```

بمجرد تنفيذ الامر RemoveHandler السابق، فلن يتم استدعاء الإجراء PersonHasDied() حتى لو تم إطلاق الحدث Die ملايين المرات.

لا يقتصر الاختلاف في قنص الأحداث بين WithEvents و AddHandler في مسألة أن الأولى وقت التصميم والثانية وقت التنفيذ، بل يتعدى الأمر أكثر من ذلك، إذ أن عملية قنص الأحداث باستخدام WithEvents يكون موجه إلى متغير الكائن بشكل حصري، أما قنص الأحداث باستخدام AddHandler فهو موجه إلى الكائن نفسه وليس المتغير الذي يشير إليه. قد يبدو لي أن الفرق لم يتضح لك بعد، لذلك سأحاول توضيح الفرق باستخدام أكثر من مؤشر إلى نفس الكائن، والبدء سنكون مع WithEvents:



```
Module Module1
    Dim WithEvents Turki As New PersonClass()

    Sub Main()
        ' Turki و Turki2 يشيران إلى نفس الكائن
        Dim Turki2 As PersonClass = Turki

        ' سيتم تنفيذ الإجراء
        ' PersonHasDied
        Turki2.KillHim()

        ' الغاء المؤشر الاول (وهو الذي عرف بـ WithEvents)
        Turki = Nothing

        ' لن يتم تنفيذ الإجراء
        ' PersonHasDied
        Turki2.KillHim()
    End Sub

    Sub PersonHasDied() Handles Turki.Die
        ArabicConsole.WriteLine("لقد توفي الشخص")
    End Sub
End Module
```

من المثال السابق يتضح لنا أن استخدام الكلمة المحجوزة WithEvents لقنص أحداث الكائن مرتبطة ارتباطاً كاملاً بمتغير (مؤشر) الكائن الذي استخدم لحظة التعريف، والدليل أننا بعدما ألغينا متغير الكائن الذي يستخدم WithEvents، لم تتم عملية تنفيذ الإجراء PersonHasDied() رغم أن الحدث Die قد تم إطلاقه بالفعل.

من ناحية أخرى، قنص الأحداث باستخدام AddHandler لا يتعامل مع مؤشر الكائن (متغير الكائن)، وإنما يتعامل مع الكائن نفسه (الذي يشير إليه المتغير)، فلو حاولت إعادة تطبيق الشيفرة السابقة ولكن باستخدام AddHandler:



```
Module Module1
    Dim Turki As New PersonClass()

    Sub Main()
        AddHandler Turki.Die, AddressOf PersonHasDied

        ' Turki و Turki2 يشيران إلى نفس الكائن
        Dim Turki2 As PersonClass = Turki

        ' سيتم تنفيذ الإجراء
        ' PersonHasDied
        Turki.KillHim()

        ' الغاء المؤشر الاول
        Turki = Nothing

        ' سيتم تنفيذ الإجراء
        ' PersonHasDied أيضا
        Turki2.KillHim()
    End Sub

    Sub PersonHasDied()
        ArabicConsole.WriteLine("لقد توفي الشخص")
    End Sub
End Module
```

سنكتشف أن الإجراء PersonHasDied() تم استدعائه لحظة انطلاق الحدث Die مرتين بالرغم من اختفاء مؤشر الكائن الأول له، لذلك سيستمر الإجراء PersonHasDied() في عملية قنص الحدث حتى موت الكائن نهائيا واختفاء جميع المؤشرات التي تشير إليه (أذكر مرة أخرى، سأحدث عن موت الكائنات بالتفصيل الممل قريبا في هذا الفصل).

ميزة أخرى في عملية قنص الأحداث باستخدام AddHandles غير ممكنة في WithEvents وهي عند التعامل مع المصفوفات، فلو حاولت استخدام WithEvents مصفوفة:

```
غير ممكن
Dim WithEvents Turki(9) As PersonClass
```

سيظهر لك مترجم اللغة رسالة خطأ والسبب ان الكلمة المحجوزة WithEvents لا يمكن تطبيقها على المصفوفات، الشيء الذي يضطرك إلى استخدام AddHandles:

```
Module Module1
    Dim Turki(5) As PersonClass

    Sub Main()
        Dim counter As Integer
        Dim Turki(5) As PersonClass

        For counter = 0 To UBound(Turki)
            Turki(counter) = New PersonClass()
            ' ممكن قمس الحدث '
            AddHandler Turki(counter).Die, AddressOf PersonHasDied
        Next

        Turki(0).KillHim()
        Turki(5).KillHim()
    End Sub

    Sub PersonHasDied()
        ArabicConsole.WriteLine("لقد توفي الشخص")
    End Sub
End Module
```

استخدام الكائنات

تعرفنا في الصفحات السابقة على الفئات وطريقة بنائها من منطلق انك مؤلف الفئة، أما الآن سنذهب إلى الجهة المقابلة ونستخدم هذه الفئة لننشئ بها كائنات ونتحدث من منطلق انك مستخدم لهذه الفئة، البداية ستكون مع توضيح حقيقة الكائن.

ما هي حقيقة الكائن؟

حديثي هنا محصور حول الكائنات من النوع المرجعي Reference Type، فكما قلت سابقا ان المتغيرات المنشأة من الفئات Classes، أو المصفوفات Arrays، أو الحروف Strings هي متغيرات من النوع المرجعي، وعليك أن تعلم علم اليقين أن الكائنات من النوع المرجعي لا تتعامل معها بشكل مباشر في شيفراتك المصدرية، حيث انك تصل إليها عن طريق متغير معرف منها يسمى مؤشر Pointer، فالمتغير Turki التالي:

```
Dim Turki As New PersonClass
```

لا يحتوي على البيانات الحقيقية للكائن، فهو ليس سوى مؤشر إلى كائن في منطقة من الذاكرة تسمى Managed Heap تدار عن طريق إطار عمل NET. وحجم المؤشر Turki سيكون دائما 4 بايت بغض النظر عن نوع الفئة التي أنشئ منها.

عندما نتعامل مع المؤشرات عليك تغيير منطقك البرمجي فهي ليست كالمتغيرات التقليدية، حيث أن المتغيرات التقليدية تحمل القيمة الفعلية دائما بينما المؤشرات لا تحمل إلا قيمة تمثل موقع البيانات الحقيقية للكائن في الذاكرة، اقلب الصفحة وادرس الشيفرة التي بأعلىها:

```
Class PersonClass
...
Public Age As Integer
...
End Class

...
...

Dim Turki As New PersonClass()
Dim Ali As New PersonClass()

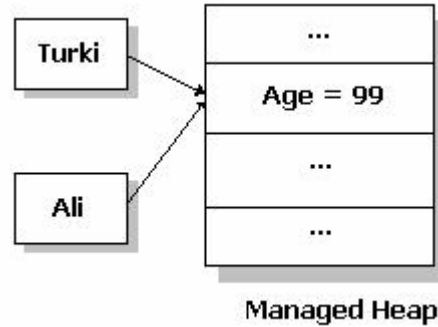
Ali.Age = 25

' الان كلاهما يشيران إلى نفس الكائن '
Ali = Turki

Turki.Age = 99

ArabicConsole.WriteLine(Ali.Age) ' 99
```

من المثال السابق يتضح لنا أن Turki و Ali مؤشرين يشيران إلى نفس الكائن، والدليل أنني عندما قمت بتغيير قيمة الحقل Age في المؤشر Turki، تأثر نفس الحقل عندما وصلت إليه عن طريق المؤشر Ali، تجد توضيح المثال السابق في (الشكل 3-1 في أعلى الصفحة التالية).

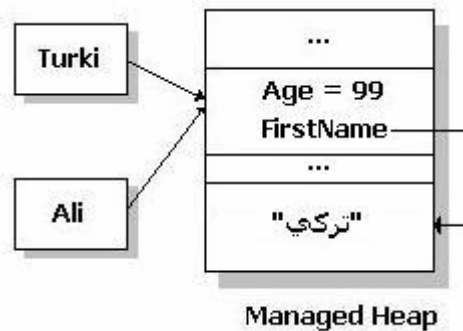


شكل 3-1: كلا المؤشران Turki و Ali يشيران إلى نفس الكائن.

دعني اجعل الأمور أكثر تعقيدا ونضيف حقل آخر من النوع String، التعامل مع هذا الحقل هو نفس التعامل مع متغيرات الفئات Classes، فالمتغيرات من النوع String ومتغيرات الفئات - كما قلت - هما من النوع المرجعي Reference Type:

```
Turki.FirstName = "تركي"
```

عليك معرفة أن الحقل FirstName ما هو إلا مؤشر أيضا، والقيمة التي يحملها تمثل بيانات لكائن وهي مستقلة في القسم Managed Heap (شكل 3-2).



شكل 3-2: الحقل FirstName ما هو إلا مؤشر آخر يشير إلى كائن مستقل آخر.

أردت من الشكل السابق أن أوضح لك بان المؤشرين Turki و Ali يشيران إلى بيانات الكائن والتي تحتوي على مؤشر FirstName آخر يشير بدوره إلى بيانات كائن آخر. برمجياً، ستستخدم المؤشر Turki الذي يشير إلى المؤشر FirstName لتصل إلى بيانات الكائن (والذي يحتوي القيمة الفعلية "تركي").

عبارات خاصة بالكائنات

حتى تتعامل مع المؤشرات أو الكائنات بطرق صحيحة، من الضروري استيعاب بعض الكلمات المحجوزة والعبارات التي تتعلق بالكائنات بشكل مباشر أو غير مباشر. البداية ستكون مع الكلمة المحجوزة New.

الكلمة المحجوزة New:

الكلمة المحجوزة New تستخدم للقيام بعملية الإنشاء الفعلي للكائن وحجز منطقة له في ذاكرة Managed Heap، يمكنك استخدامها لحظة التصريح عن المتغير (المؤشر) أو عند إسناد قيمة له بمعامل المساواة "=":

```
' إنشاء الكائن لحظة التصريح عن المتغير '
Dim TestObject As New TestClass ()
```

```
' احذف الاقواس التي قد يضيفها المحرر '
Dim TestObject As TestClass
' تم إنشاء الكائن الان '
TestObject = New TestClass
```

مع ذلك، لا يمكنك استخدام New لحظة التصريح عن متغير إن كان احد حقول Fields التركيبات من النوع Structures:

```
Structure TestStructure
    Dim TestObject As New TestClass () ' رسالة خطأ
End Structure
```

الكلمة المحجوزة New مرنة جداً بحيث يمكنك استخدامها في أي مكان تقريباً من شيفراتك المصدرية، فيمكنك مثلاً إنشاء الكائن لحظة إرساله إلى إجراء:

```

Sub Main()
    ' إنشاء الكائن لحظة ارسال الوسيطة
    TestSub(New TestClass())
    '
End Sub

Sub TestSub(ByVal obj As TestClass)
    obj.DoMethod ()
    '
End Sub

```

سيتم تنفيذ المشيدات مباشرة بعد إنشاء نسخة جديدة من الكائن باستخدام New، لذلك إن كان للفئة مشيدات، فلا بد من إرسال وسيطات المشيدات لحظة استخدام New، وإن لم تكن للفئة أي مشيدات فيمكنك الاكتفاء بوضع الأقواس خالية:

```

' لا توجد وسيطات لمشيد الفئة
Dim TestObject As New TestClass ()

' يحتوي مشيد الفئة على وسيطين
Dim AnotherObject As New AnotherClass(2, 5)

```

التركيب With ... End With

يستخدم التركيب With ... End With لتسريع عملية الوصول إلى أعضاء الكائن دون الحاجة لكتابة اسم الكائن، فبدلاً من الوصول إلى الأعضاء بهذا الشكل:

```

Dim TestObject As New TestClass()
...
...
TestObject.Field1 = 10
TestObject.Field2 = 20
TestObject.Method1 (10)
TestObject.Property1 (2, 2) = 10

```

يمكنك استخدام With ... End With بهذه الطريقة:

```

Dim TestObject As New TestClass()
...
...
With TestObject
    .Field1 = 10
    .Field2 = 20
    .Method1 (10)
    .Property1 (2, 2) = 10
End With

```

القيمة Nothing:

إسناد القيمة Nothing إلى المؤشر تؤدي إلى إلغاء العلاقة بين ذلك المؤشر والكائن الذي يشير له، وضع في عين الاعتبار أن المؤشر الذي لا يشير إلى الكائن لا يمكنك الوصول إلى أعضائه:

```
Dim Turki As New PersonClass()
Turki.Name = "تركي العسيري"
Turki = Nothing
Turki.Age = 99 ' رسالة خطأ
```

نقطة هامة جداً: إسناد القيمة Nothing للمؤشر لا تؤدي إلى موت الكائن الذي كان يشير إليه وإلغائه من الذاكرة، فالمسألة تحتاج إلى تفصيل تجده قريباً في فقرة **حياة وموت الكائنات**.

المعامل Is:

استخدم المعامل Is مع المؤشرات إن كنت تود التحقق من أن المؤشر يشير إلى نفس الكائن:

```
Dim Turki As New PersonClass()
Dim Khaled As New PersonClass()

ArabicConsole.WriteLine(Turki Is Khaled) ' False

' المؤشران يشيران إلى نفس الكائن الآن
Khaled = Turki

ArabicConsole.WriteLine(Turki Is Khaled) ' True
```

كما تلاحظ، القيمة التي يعود بها المعامل Is منطقية (من النوع Boolean) مما يجعلها مناسبة جداً لاستخدامها داخل جملة شريطة:

```
If Not Turki Is Khaled Then
    Turki = Khaled
End If
```

يمكنك الاستفادة أيضاً من المعامل Is لمعرفة ما إذا كان المؤشر يشير حقاً إلى كائن أم لا، وذلك باستخدامه مع القيمة Nothing:

```
' هل المؤشر Turki لا يشير إلى كائن؟
If Turki Is Nothing Then
    Turki = New PersonClass()
End If
```

العبارة ... Is TypeOf:

العبارة ... Is TypeOf شبيهة إلى حد كبير مع المعامل Is السابق، ولكنها تستخدم للتحقق من نوع الفئة التي يشير إليها متغير الكائن:

```
' هل الكائن Turki منشئ من الفئة PersonClass ؟
If TypeOf Turki Is PersonClass Then
    Turki.Name = "تركي العسيري"
End If
```

الكائن Me:

يمكنك استخدام الكائن Me داخل الفئة للتعبير عن الكائن الحالي، بعبارة أخرى الكائن Me هو مؤشر للنسخة الحالية من الكائن، فعندما أقول: أنا، فإنني أقصد نفسي وعندما تقول أنت: أنا، فإنك تقصد نفسك:

```
Class PersonClass
    Public Name As String

    Sub PrintName ()
        ' طباعة قيمة الحقن Name للكائن الحالي
        ArabicConsole.WriteLine ( Me.Name )
    End Sub
    ...
End Class
```

صحيح أن الكائن Me يمثل مؤشر الكائن الحالي، إلا أنه يختلف في بنيته التحتية ومسائل تقنية أخرى عن مؤشرات (متغيرات) الكائنات التقليدية. فمثلاً، عن طريق Me تستطيع الوصول إلى الأعضاء من النوع Private، ولكنك لن تستطيع إسناد أي قيمة إلى Me:

```
Class PersonClass
    Private Name As String

    Sub PrintName ()
        ArabicConsole.WriteLine (Me.Name) ' ممكن
        Me = Nothing ' رسالة خطأ
    End Sub
    ...
End Class
```

إسناد القيم

استخدام معامل إسناد القيم "=" بين متغيرات الكائنات لا يقوم بنسخ بيانات الكائن، وإنما زيادة عدد المؤشرات التي تشير إلى الكائن -كما أخبرتك أكثر من مرة سابقاً:

```
Dim X As New ClassA
Dim Y As ClassA

Y = X
```

في حالة اختلاف نوع الفئات التي تشير لها الكائنات، فلن تستطيع إسناد القيم إليها بنسخ مؤشراتهما:

```
Dim X As New ClassA
Dim Y As ClassB

' خطأ لاختلاف نوع فئات الكائنات
Y = X
```

كذلك الحال عند استخدام الكلمة المحجوزة New لحظة إسناد القيم، فلا بد من توافق نوع الفئة مع نوع المتغير الذي أعلنت عنه:

```
Dim X As ClassA

X = New ClassA () ' ممكن
X = New ClassB () ' رسالة خطأ
```

مع ذلك تستطيع إنشاء الكائنات باستخدام New أو إسناد القيم لها دون الحاجة لتوافق نوع الفئة مع المتغير، وذلك في حالة تصريحك للمتغيرات من النوع Object:

```
Dim X As Object
Dim Y As Object

' إنشاء كائن من الفئة ClassA
X = New ClassA ()

' ممكن
Y = X
Y.MethodInClassA ()

' الكائن أصبح منشئ من الفئة ClassB
Y = New ClassB ()
```

```
' ممكن
X = Y
X.MethodInClassB ()
```

رغم أن تصريح المتغيرات بالنوع Object يعطي مرونة كبيرة عند إسناد القيم لها أو إنشاء الكائنات منها، إلا أن ذلك يسبب الكثير من البطء في عملية التنفيذ، حيث تتطلب من مترجم Visual Basic .NET تحويل نوع بيانات المتغير والتحقق من كافة الطرق والخصائص الموجودة في الكائن قبل استدعائها والبحث أيضا عن مواقعها. وحتى لا أتوغل في التفاصيل التقنية أكثر، دعني أخبرك أن هذه الطريقة تسمى **الربط المتأخر Late Binding**، وجميع الطرق التي ذكرتها سابقا في هذا الفصل هي **الربط المبكر Early Binding** وهو أسرع بعشرات -إن لم يكن مئات أحيانا- المرات من الربط المتأخر:

```
' الربط المبكر
Dim X As TestClass = New TestClass ()

' الربط المتأخر
Dim X As Object = New TestClass ()
```

العبارة Option Strict مرة أخرى:

عند تفعيلك للعبارة Option Strict On في احد ملفات المشروع، سيمنعك Visual Basic .NET من استخدام أسلوب الربط المتأخر Late Binding:

```
' رسالة خطأ في حالة تفعيلك للعبارة
' Option Strict On
Dim X As Object

X = New ClassA () ' ليس هنا السبب

X.MethodInClassA() ' هنا السبب
```

أخي الكريم، عليك الانتباه عن أن سبب الخطأ ليس بسبب عملية الإسناد السابقة، وإنما بسبب استدعاء الطريقة MethodInClassA() عن طريق الربط المتأخر، قد تستغرب مدى الجدوى من المتغير X السابق ما دمت لن تستطيع استدعاء أعضائه، ولكن في اغلب الأحوال يلجأ المبرمجون إلى هذه العملية لجعل وسيلة الإجراء Parameter تستقبل قيم مهما كان نوعها (كما هو الحال مع الكائن ArabicConsole)، كما تفيدهم هذه الطريقة أيضا لإبقاء الكائن على قيد الحياة والاحتفاظ بمتغير يشير إليه حتى تتمكن من إعادة إسناده إلى متغير آخر من نفس الفئة:

```
Dim X As Object
Dim Y As ClassA
...
...

X = New ClassA ()

Y = X ' عملية الإسناد
```

مع ذلك، سيظهر مترجم .NET Visual Basic رسالة خطأ بسبب تعارض أنواع المتغيرات، فالمتغير X السابق من النوع Object (رغم أن القيمة التي يحتفظ بها من النوع ClassA) بينما المتغير Y من النوع ClassA، والحل الوحيد الذي يمكنك من إنجاز هذه العملية هو باستخدام معامل التحويل CType والذي يتطلب القيمة ومن ثم النوع المراد تحويله إليها:

```
Dim X As Object
Dim Y As ClassA
...
...

X = New ClassA ()

Y = CType (X, ClassA) ' لابد من استخدام المعامل CType

Y.MethodInClassA() ' توكل على الله واستدعي الطرق كما تريد
...
...
```

ملاحظة

التعبير "المعامل CType" ليس خطأ مطبعي، لأن CType هو معامل Operator وليست دالة Function كدوال التحويل CLng(), CInt(), الخ....، فالمعامل CType هو احد سمات لغة البرمجة Visual Basic .NET. مع ذلك، يمكنك استخدام الدالة CType احمر احمر اقصد المعامل CType ليعمل عمل دوال التحويل الأخرى:

```
Dim Y As Integer = CInt ("10")
Dim Y As Integer = CType ("10", Integer)
```

حياة وموت الكائنات

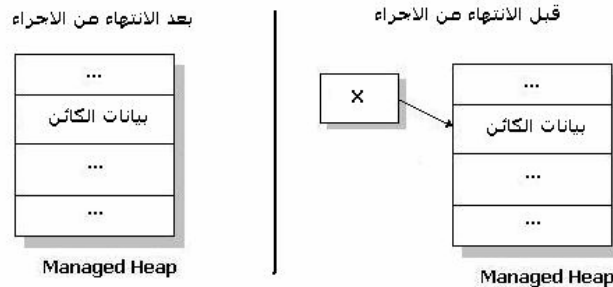
إن تحدثنا عن المتغيرات ذات القيمة Value Type، فهي ستبقى محتفظة بقيمتها استنادا لعمرها Lifetime كما ذكرت في الفصل السابق، فلو كان لدينا المتغير x التالي:

```
Sub MySub ()
    Dim X As Integer = 10
    ...
End Sub
```

سيبقى هذا المتغير محتفظا بقيمته حتى الخروج من الإجراء ويختفي ويزال نهائيا من الذاكرة. أما الحديث عن المتغيرات المرجعية Reference Type:

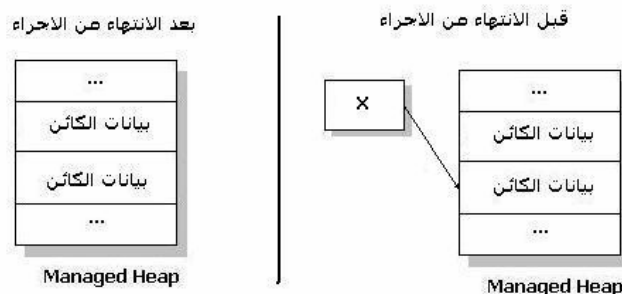
```
Sub MySub ()
    Dim X As New SimpleClass
    ...
End Sub
```

فسيبقى المؤشر X محتفظا بقيمته (القيمة هنا عنوان بيانات الكائن في ذاكرة Managed Heap) حتى نهاية الإجراء ومن ثم سيزال المؤشر X من الذاكرة وتلغى قيمته ولكن الكائن سيستمر في الذاكرة! (شكل 3-3).



شكل 3-3: الكائن ما زال موجود في ذاكرة Managed Heap

ازيدك من الشعر بيت، هل تصدق انك لو قمت باستدعاء الإجراء MySub() السابق مرة أخرى سيتم إنشاء نسخة جديدة من الكائن وستبقى حتى بعد نهاية الإجراء إلى جانب النسخة السابقة (شكل 4-3 بالصفاة المقابلة).



شكل 3-4: نسخة أخرى من الكائن في ذاكرة Managed Heap لم يتم ازلتها أيضا.

عجبا على هذا الغباء! لماذا لا تزال الكائنات من الذاكرة رغم أننا لا نريدها ولن نستطيع الوصول لها بسبب فقدان مؤشراتهما؟ الجواب يتعلق بالبنية التحتية لإطار عمل .NET Framework وطريقة إدارته للذاكرة Managed Heap وتعامله معها. وحتى تستوعب ذلك، عليك أن تتعرف -شئت أم أبيت- على المجموعة Garbage Collection.

المرجعية الدائرية Circular Reference:

قبل التوغل في تفاصيل المجموعة Garbage Collection بودي توضيح قضية تعتبر معاناة كبيرة واجهها المبرمجون -واجهتها أنا شخصيا- في لغات البرمجة السابقة، ألا وهي **المرجعية الدائرية Circular Reference**. في السابق، كانت الكائنات تموت مباشرة بعد اختفاء جميع المؤشرات التي تشير لها ومن ثم تتم عملية تفريغ مساحتها من الذاكرة، بعبارة أخرى، عندما يصل عدد المؤشرات التي تشير إلى الكائن (يسمى **العداد Counter**) إلى صفر، تتم عملية إلغاء بيانات الكائن الفعلية من الذاكرة.

وبما أن برامجنا كانت تعتمد اعتمادا كبيرا جدا على الكائنات، ظهرت لنا مشكلة المرجعية الدائرية Circular Reference والتي تبقي كائنين على قيد الحياة رغم اختفاء جميع مؤشراتهما. (إن كنت من المبرمجين المبتدئين، فاعذرني على ذكر هذه المسائل ولكنها ستعرفك على أحد الأسباب التي حدثت بمطوري .NET. أن يعتمدوا على المجموعة Garbage Collection)، وحتى تفهم كيف تحدث مشكلة المرجعية الدائرية افترض أن لدينا هذه الفئة والمسطورة في أعلى الصفحة التالية:

```

Class PersonClass
    Public Name As String
    Public Brother As PersonClass
End Class

```

سننشئ كائنين ونسند اليهما قيم للحقول:

```

Dim Turki As New PersonClass()
Dim Abdullah As New PersonClass()

Turki.Name = "تركي العسيري"
Abdullah.Name = "عبدالله العسيري"

```

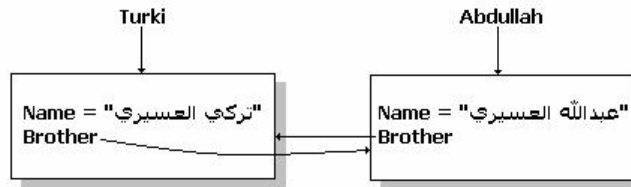
حتى الآن الكلام جميل جدا ولا توجد به أي مشاكل، ولكن عندما تسند القيم إلى الحقل Brother ستبدأ رحلة العناء والشقاء مع عالم الكائنات:

```

Turki.Brother = Abdullah
Abdullah.Brother = Turki

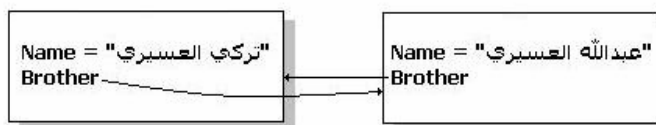
```

في السطر السابق قمنا بربط الكائنين Turki و Abdullah بحيث يشاران إلى بعضهما البعض (شكل 3-5)، هذه العملية تسمى المرجعية الدائرية Circular Reference.



شكل 3-5: المرجعية الدائرية Circular Reference.

والآن ركز معي يا عسل، تخيل أن الشيفرات السابقة كانت داخل إجراء معين وانتهى الإجراء بعد إسناد القيم، منطقياً ستفقد المؤشرات Turki و Abdullah قيمها وتزال من الذاكرة، ولكن الكائنات ما زالت موجودة وذلك بسبب الحقل Brother الذي لا يزال يشير إلى الكائن ويحميه من الموت (شكل 3-6) وهذه هي مشكلة المرجعية الدائرية.



شكل 3-6: مشكلة المرجعية الدائرية Circular Reference حدثت.

كما ترى في (الشكل 3-6)، الكائنات ستستمر بالذاكرة، ويمكنك التأكد بنفسك باستخدام هذه الشيفرة:

```

Dim Turki As New PersonClass()
Dim Abdullah As New PersonClass()

Turki.Name = "تركي العسيري"
Abdullah.Name = "عبدالله العسيري"

Turki.Brother = Abdullah
Abdullah.Brother = Turki

Turki = Nothing ' قتل الكائن الاول

' الكائن الاول لا يزال حيا يرزق
ArabicConsole.WriteLine(Abdullah.Brother.Name) ' تركي العسيري
  
```

قد تتباهى بذكائك وتعطيني حلا سريعا لمشكلة المرجعية الدائرية، وذلك بإلغاء جميع الحقول المسببة لهذه المشكلة قبل الخروج من الإجراء ثم إلغاء المؤشرات الرئيسية:

```

' لنلغي الحقول اولا
Turki.Brother = Nothing
Abdullah.Brother = Nothing

' ثم المؤشرات الرئيسية
Turki = Nothing
Abdullah = Nothing
  
```

كلامك صحيح ولا غبار عليه، ولكنه ليس عملياً إلا في مثالنا السابق فقط، ففي البرامج الكبيرة سنتعامل مع مئات الكائنات، وكل كائن يحتوي على عشرات الحقول والتي تشير إلى كائنات أخرى لتصبح إمكانية نسيان تنظيف احد الحقول مسألة لا إرادية. وصدقني، مشكلة المرجعية الدائرية يقع فيها كبار المبرمجين المحترفين والدليل هو استهلاك البرامج للمصادر الكبيرة من الذاكرة والتي لا تستخدم.

المجموعة Garbage Collection:

بعدما عرضت عليك مشكلة المرجعية الدائرية، قد تفضل العودة إلى البرمجة الإجرائية التقليدية، ولكن أريدك أن تتغلب على الخوف الذي يصيبك من هذه المعلومات وتصبح مبرمج شجاع (كلمة الشجاعة لا يشترط ربطها ببرمجة API Windows J) وتستمر في استخدام الكائنات، لأنه يسرني إخبارك أن مشكلة المرجعية الدائرية لن تصادفها أبداً وذلك بفضل المجموعة Garbage Collection.

حسناً، يوفر لك إطار عمل .NET Framework المجموعة Garbage Collection والهدف الرئيسي منها هو تفريغ الذاكرة من الكائنات الغير مرغوبة، واعني بالكائنات الغير مرغوبة هي الكائنات التي لا يشير إليها أي مؤشر أو متغير ظاهر في شيفرات البرنامج وليس جميع المؤشرات، فلو أمعنت النظر في (الشكل 3-6) ستلاحظ ان الكائنين لهما مؤشران باسم Brother، وبما أن هذا المؤشر ليس ظاهر في شيفراتك المصدريّة، ستقوم المجموعة Garbage Collection بإلغاء الكائنات وتحرير مساحتها من الذاكرة.

كلام مريح جداً ويزيح عن المبرمج هم التفكير في مسألة تحرير كائناته من الذاكرة، ولكن متى ستتم هذه العملية؟ إن كان السؤال موجه لي شخصياً، فانا لا اعلم متى ستتم عملية التحرير الفعلي للذاكرة، حيث انك في كل مرة تنشئ فيها كائن جديد، سيستمر هذا الكائن في ذاكرة Managed Heap حتى تمتلئ تماماً، وبعد ذلك تتم عملية التحرير تلقائياً من قبل المجموعة Garbage Collection، وان كانت الكائنات المستخدمة قليلة ولا تستهلك تلك المساحة الكبيرة، فعملية التحرير لن تتم حتى نهاية البرنامج.

مع ذلك، يمكنك طلب عملية التحرير من المجموعة Garbage Collection في أي وقت تريده يدوياً باستدعاء الطريقة Collect() التابعة للكائن GC:

```
GC.Collect()
GC.WaitForPendingFinalizers()
```

الكائن GC معرف في إطار عمل .NET Framework. وهو يمثل المجموعة Garbage Collection، وبالنسبة للطريقة WaitForPendingFinalizers() فهي توقف عملية تنفيذ البرنامج مؤقتاً حتى الانتهاء من عملية تحرير المساحة.

مع ذلك، أنصحك بشدة عدم استدعاء الطريقة GC.Collect() بنفسك يدوياً، فعملية تحرير الذاكرة تتطلب وقت طويل -خاصة ان كانت الكائنات كثيرة- مما يؤدي إلى بطء ملحوظ في تنفيذ

البرنامج، لذلك دع إطار عمل NET Framework يحدد الوقت المناسب للقيام بهذه العملية، واطلع منها أنت!

ملاحظة

ذكرت قبل قليل أن الطريقة () WaitForPendingFinalizers توقف عملية تنفيذ البرنامج بشكل مؤقت، ولكن الحقيقة أنها توقف مسار التنفيذ Thread الحالي فقط. سأحدث بالتفصيل الممل عن مسارات التنفيذ Threading في الفصل العاشر **مسارات التنفيذ** Threading.

الموت المنطقي والموت الحقيقي للكائنات:

ببساطة شديدة، الموت المنطقي للكائنات يحدث عندما تختفي جميع المؤشرات التي تشير إلى الكائن، أما الموت الحقيقي للكائن فيحدث عندما تقوم المجموعة Garbage Collection بتحرير مساحته من الذاكرة Managed Heap، فالكائن التالي:

```
Sub Test ()
    Dim TestObject As New TestClass()
    ...
End Sub
```

سيموت منطقياً عند نهاية الإجراء وذلك لاختفاء المؤشر TestObject الذي يشير له، ولكنه لن يموت بشكل حقيقي إلا إن تمت عملية التحرير من قبل المجموعة Garbage Collection. كيف يمكننا معرفة الوقت الذي يموت فيه الكائن، وذلك لأن لغات البرمجة القديمة كانت توفر إجراء يسمى **المهدم Destructor** يتم استدعائه لحظة موت الكائن. يمكنك Visual Basic .NET من تعريف الإجراء Finalize() والذي يتم استدعائه من قبل المجموعة Garbage Collection (أي يتم تنفيذه لحظة الموت الحقيقي وليس المنطقي):

```
Class TestClass
    ...
    Protected Overrides Sub Finalize()
        ' اكتب الشيفرة هنا
    ...
End Sub
End Class
```

انظر أيضا

استخدمت محدد الوصول المحمي Protected والمعامل إعادة القيادة Overrides وذلك لأنني قمت باشتقاق الإجراء Finalize() من الكائن System.Object، الفصل القادم **الوراثة** سيوضح لك السبب.

مع ذلك، فإن الاعتماد على الإجراء Finalize() السابق فيه شيء من الخطأ، وذلك لأننا لا نعلم بالضبط متى سيكون الموعد الحقيقي لموت الكائن، ففي لغات OOP السابقة كنا نستخدم هذا النوع من المهدمات لحظة الموت المنطقي للكائن، أما الموت الحقيقي فقد يتم بعد الموت المنطقي بثواني، دقائق، أو حتى ساعات فهو مرتبط ارتباطاً كلياً بعملية التحرير التي تقوم بها المجموعة Garbage Collection.

تنبيه

لا تحاول الوصول إلى كائنات أخرى من داخل الإجراء Finalize() (كـ ArabicConsole وغيرها) وذلك لأن هذه الكائنات قد تكون تمت عملية موتها الحقيقي وأُخليت من الذاكرة، مما قد يتسبب في ظهور رسالة خطأ. بصفة عامة، يستخدم الإجراء Finalize() للقيام بعمليات أخيرة لا تستخدم كائنات أخرى في نفس المشروع أو الكائن نفسه.

بما أننا لا نستطيع تعريف مهدم في الفئة يتم تنفيذه لحظة الموت المنطقي للكائن، اعتمد مبرمجوا .NET على محاكاة المهدم وتعريف إجراء باسم Dispose() في الفئة:

```
Class TestClass
    Implements IDisposable

    Public Sub Dispose() Implements System.IDisposable.Dispose

    End Sub
    ...
    ...
End Class
```

انظر أيضا

الكلمة المحجوزة Implements تتعلق بالواجهات Interfaces وهو موضوع الفصل الخامس الواجهات، التفويض، والمواصفات.

على المبرمجين الذين ينشئون كائنات من هذه الفئة، استدعاء الطريقة Dispose() لحظة الموت المنطقي للكائن، حتى يتمكن من القيام بما عليه القيام به (كإغلاق الملفات، قطع الاتصالات مع قواعد البيانات، تحرير مصادر النظام...الخ):

```
Dim TestObject As New TestClass
```

```
...
...
```

```
' استدعي الطريقة اولا
TestObject.Dispose()
```

```
' ثم اقتل الكائن منطقيا
TestObject = Nothing
```

بكل تأكيد قد تنسى -أو ينسى أحد المبرمجين- استدعاء الطريقة Dispose() لحظة إلغاء المؤشر الأخير للكائن، لذلك لنجعل الطريقة Finalize() تستدعي Dispose() كنوع من الاحتياط:

```
Protected Overrides Sub Finalize()
    Dispose()
End Sub
```

وحتى لا يتم استدعاء الطريقة Dispose() أكثر من مرة بالخطأ، يفضل استخدام متغير سناتيكي لمنع ذلك:

```
Public Sub Dispose() Implements System.IDisposable.Dispose
    Static CancelDisposing As Boolean

    If CancelDisposing Then
        Exit Sub
    End If
    ...
    ...
    CancelDisposing = True
End Sub
```

أسلوب أكثر امانا لكتابة المهدمات Dispose() و Finalize():

ذكرت سابقا، أن المبرمج يستدعي الطريقة Dispose() قبل لحظة موت الكائن منطقيا، مع ذلك لدى المبرمج فرصة كبيرة لاستدعاء طرق وخصائص الكائن بعد موته منطقيا (وهي فرصة كفيلة بحدوث عشرات المشاكل والشوائب البرمجية):

```
Dim TestObject As New TestClass

...

...

' قام المبرمج باستدعاء المهدم
TestObject.Dispose()

' لديه فرصة لاستدعاء إجراءات الكائن
' بعد موته المنطقي
TestObject.MethodInClass ()
```

لذلك عليك منع المبرمج من استدعاء طرق وخصائص الكائن بعد استدعاء المهدم Dispose()، قد تفعل ذلك باستخدام العبارة Exit Sub:

```
Class TestClass
...
...
Private CancelDisposing As Boolean

Sub MethodInClass ()
    ' منع المبرمج من استدعاء الإجراءات بعد
    ' Dispose() المهدم
    If CancelDisposing Then
        Exit Sub
    End If
    ...
    ...
End Sub

Public Sub Dispose() Implements System.IDisposable.Dispose
    If CancelDisposing Then
        Exit Sub
    End If
    ...
    ...
    CancelDisposing = True
End Sub
End Class
```

أو رمي استثناء **Throw an Exception** كما تفعل سائر كائنات إطار عمل .NET. الأخرى:

```
Class TestClass
...
Sub MethodInClass ()
    ' منع المبرمج من استدعاء الإجراء بعد
    ' Dispose() المهدم
    If CancelDisposing Then
        Throw New ObjectDisposedException("TestClass")
    End If
...
End Sub
...
End Class
```

انظر أيضا

رمي استثناء Throw an Exception هي عملية أحداث خطأ وقت التنفيذ Run Time Error في البرنامج، الفصل السابع **اكتشاف الأخطاء** خاص بتدارك ورمي الاستثناءات.

المزيد أيضا، قد تستدعي المهدم Dispose() من داخل المهدم Finalize() كنوع من الاحتياط - كما ذكرت - وذلك في حالة عدم استدعائه من قبل المبرمجين لحظة الموت المنطقي للكائن:

```
Protected Overrides Sub Finalize()
    Dispose()
End Sub
```


مشكلة أخرى -ليست كبيرة- في الأسلوب السابق، وهي أن الإجراء Finalize() سيتم استدعائه دائما من قبل المجموعة Garbage Collection حتى وإن قام المبرمج باستدعاء المهدم Dispose() يدويا، ولا تنسى أن عملية تحرير المصادر من قبل المجموعة Garbage Collection تستغرق وقت طويل دون حاجة، مما يعني أننا لسنا بحاجة لاستدعاء الإجراء Finalize() من قبل المجموعة Garbage Collection إن تم استدعاء الإجراء Dispose() من

قبل المبرمج. لذلك، قد تستدعي الطريقة GC.SuppressFinalize() من داخل الإجراء Dispose() لمنع المجموعة Garbage Collection من استدعاء المهدم Finalize():

```
Class TestClass
...
...
Public Sub Dispose() Implements System.IDisposable.Dispose
...
...
' منع المجموعة Garbage Collection
' Finalize من استدعاء المهدم
GC.SuppressFinalize(Me)
End Sub
End Class
```

نقطة أخرى حول الشيفرة قبل السابقة، قد تبدو فكرة استدعاء المهدم Dispose() من داخل المهدم Finalize() ذكية، ولكن توجد بها مشكلة قد تسبب في أحداث كوارث في البرنامج (خصوصاً مع الفئات الكبيرة والكائنات المعقدة). قد تحدث هذه المشكلة لحظة الموت الحقيقي للكائن، حيث سيتم استدعاء الإجراء Dispose() والذي قد تصل شيفراته المصدرية إلى كائنات أخرى ممتدة بسبب المجموعة Garbage Collection.

يمكنك الالتفاف حول هذه المشكلة بإعادة تعريف Overloads المهدم Dispose() بحيث يستقبل وسيطة إضافية تحدد فيها ما إذا تم استدعاء المهدم من قبل المبرمج (لحظة الموت المنطقي) أو من قبل المهدم Finalize() (لحظة الموت الحقيقي)، وبذلك يكون الأسلوب الأكثر اماناً لكتابة المهدمات Finalize() و Dispose() (والذي تتبعه جميع كائنات إطار عمل .NET Framework) كالتالي:

```

Class TestClass
Implements IDisposable
...
Private CancelDisposing As Boolean

Public Overloads Sub Dispose() Implements IDisposable.Dispose
Dispose(True)
GC.SuppressFinalize(Me)
End Sub
```

```
Private Overloads Sub Dispose(ByVal disposing As Boolean)
    If CancelDisposing Then
        Exit Sub
    End If

    If disposing Then
        ' الشيفرات التي يتم تنفيذها لحظة استدعاء
        ' الإجراء من قبل المراجع (الموت المنطقي)
    End If

    ' الشيفرات التي يتم تنفيذها لحظة استدعاء
    ' الإجراء من قبل المهمل (Finalize ())
    ' (الموت الحقيقي)

    CancelDisposing = True
End Sub

Protected Overrides Sub Finalize()
    Dispose(False)
End Sub
End Class
```

إرسال الكائن بالمرجع أو القيمة

تحدثت سابقاً عن عملية إرسال الكائن إلى وسيطات الإجراءات بالمرجع أو بالقيمة، وذكرت أنه في الحالتين سيتمكن الإجراء من تعديل قيم أعضاء الكائن المرسل. أما الفرق بين استخدام ByVal و ByRef عند وسيطات الإجراءات فهو تقني بحت، إذ إن إرسال متغير الكائن باستخدام ByVal سيؤدي إلى نسخ قيمة المؤشر إلى مؤشر آخر، مما يزيد عدد المؤشرات التي تشير إلى نفس الكائن. أما إرسال متغير الكائن باستخدام ByRef، فهي ترسل عنوان ذلك المؤشر وليس قيمة المؤشر (والتي تحمل عنوان الكائن نفسه)، المثال التالي قد يوضح أحد الفروق:

```
' لا تؤدي إلى موت الكائن أبدا
Sub ByValSub(ByVal TestObject As TestClass)
    TestObject = Nothing
End Sub

' تؤدي إلى موت الكائن إن أرسل
' المؤشر الوحيد
Sub ByRefSub(ByRef TestObject As TestClass)
    TestObject = Nothing
End Sub
```

إن كنت تخشى إرسال كائناتك إلى إجراءات تستقبل وسيطات بالمرجع ByRef من أن تهلكها، فهذا من حقك ولا يلومك أي مبرمج، لذلك أضف أقواس إضافية حول القيمة المرسله لتجبر Visual Basic .NET على إرسال المتغير بالقيمة (حتى وإن كانت الوسيطة تستقبل بالمرجع):

```
Dim TestObject As New TestClass
' تم الارسال هنا بالقيمة رغم ان الوسيطات
' يتوقع انها تكون بالمرجع، فلا خوف على الكائن من الموت
ByRefSub ((TestObject))
```

الأعضاء المشتركة

كما رأيت سابقا، الأعضاء التابعة للكائنات تكون مستقلة بالكائن التابعة له، إلا أنك في حالات كثيرة تود من الكائنات المنشأة من فئة معينة أن تتشارك البيانات فيما بينها. في الفقرات التالية سألقي الضوء على الأعضاء المشتركة **Shared Members** وكيفية تطبيقها على الحقول، الطرق، الخصائص، والأحداث.

الحقول المشتركة Shared Fields

الحقول المشتركة **Shared Fields** هي حقول تكون قيمها مشتركة بين كافة الكائنات المنشأة من الفئات، وحتى تجعل الحقل مشترك استخدم الكلمة المحجوزة Shared عند التصريح عن الحقل. في الفئة التالية عرفت حقلين الأول تقليدي والثاني مشترك:

```
Class TestShared
    Public FirstName As String
    Shared Public LastName As String
End Class
```

عند التعامل مع الحقل الأول **FirstName**، سيستقل كل كائن بقيمة مختلفة لهذا الحقل، وهذا أمر منطقي:

```
Dim Meshari As New TestShared()
Dim Turki As New TestShared()

Meshari.FirstName = "مشاري"
Turki.FirstName = "تركي"

ArabicConsole.WriteLine(Meshari.FirstName) ' مشاري
ArabicConsole.WriteLine(Turki.FirstName) ' تركي
```

أما إن حاولت التعامل مع الحقل الثاني LastName، فعليك الأخذ بعين الاعتبار انه مشترك بين جميع الكائنات المنشأة من الفئة. لمعرفة الفرق في هذه الحالة، راقب الشيفرة التالية:

```
Meshari.LastName = "الفحطاني"
Turki.LastName = "العسيري"

ArabicConsole.WriteLine(Meshari.LastName)      ' العسيري
ArabicConsole.WriteLine(Turki.LastName)        ' العسيري
```

نستنتج من ذلك، ان الحقول المشتركة ما هي إلا متغيرات قابلة للوصول من جميع الكائنات المنشأة من فئة معينة، بحيث تتشارك هذه الكائنات في قيمها، لذلك تستطيع الوصول إلى الحقول المشتركة مباشرة من اسم الفئة دون الحاجة لإنشاء كائن منها:

```
' لاحظ استخدام الفئة TestShared
' وليس الكائن Turki
TestShared.LastName = "العسيري"
```

قد نقيّد الحقول المشتركة -مثلاً- في معرفة عدد الكائنات المنشأة من فئة معينة:

```
Class TestShared
    Public Shared NumOfObjects As Integer = 0

    Sub New()
        NumOfObjects += 1
    End Sub
End Class
```

ستلاحظ ان الحقل NumOfObjects مشترك بين كافة الكائنات المنشأة، وستزداد قيمته في كل مرة تنشئ كائن جديد من الفئة TestShared:

```
Dim ObjectOne As New TestShared()
Dim ObjectTwo As New TestShared()

ArabicConsole.WriteLine(ObjectOne.NumOfObjects)      ' 2

Dim ObjectThree As New TestShared()

ArabicConsole.WriteLine(ObjectThree.NumOfObjects)    ' 3
```

بل يمكنك تقليص عدد الكائنات التي تمكن مستخدم الفئة من إنشائها، فقد تظهر رسالة خطأ إن تجاوز عدد الكائنات المنشأة من الفئة عن عدد معين تحدده في جملة شرطية:

```

Sub New()
    NumOfObjects += 1

    If NumOfObjects > 10 Then
        ' الامر التالي سيؤدي إلى ظهور رسالة خطأ
        ' انتقل للفصل السابع: اكتشاف الاخطاء
        ' لمزيد من التفاصيل
        Throw New Exception()
    End If
End Sub

```

مئات التطبيقات والأفكار التي تستطيع إنجازها بفضل الحقول المشتركة، كل ما يهمني هنا توضيح الفكرة لك حتى تتمكن من عمل ما تريد، وقد تجد في فصول هذا الكتاب الكثير من الأمثلة التي تستخدم الحقول المشتركة.

الطرق المشتركة Shared Methods

الفكرة من الطرق المشتركة **Shared Methods** قد تبدو غريبة بعض الشيء، فالطرق العادية دائماً ما تكون مشتركة بين كائنات الفئات، ولكن الفرق بينهما تقني بحت، فالطرق المشتركة لا تنتمي إلى أي نسخة كائن Object Instance معينة، لذلك يمكنك استخدامها دون الحاجة لإنشاء وتعريف كائن جديد، فلو أضفنا هذه الطريقة إلى الفئة السابقة:

```

Class TestShared
    ...
    Public Shared Function CheckNumOfObjects() As Integer
        Return NumOfObjects
    End Function
End Class

```

يمكنك استدعاء الطريقة `CheckNumOfObjects()` مباشرة دون الحاجة لتعريف كائن من الفئة:

```

Dim X As TestShared

If TestShared.CheckNumOfObjects < 10 Then
    X = New TestShared()

    ArabicConsole.WriteLine(X.NumOfObjects)
End If

```

من الضروري توضيح نقطة هامة، الشيفرة الموجودة داخل الطرق المشتركة ليس لها صلاحيات كالشيفرة الموجود في الطرق العادية، المقصد من كلمة الصلاحيات في هذا السياق هو

أن الشيفرة الموجودة داخل الطرق المشتركة لا يمكن أن تستخدم الأعضاء (الحقول، الطرق، الخصائص، والأحداث) العادية والتابعة لنفس الفئة، ولكن يمكن أن تصل إلى الأعضاء المشتركة فقط. وهذا الأمر يبدو منطقياً إن علمنا أن الطرق المشتركة لا تتبع لكائن معين فكيف تريد أن تصل إلى أعضاء كائن لم يتم إنشائه:

```
' ستظهر رسالة خطأ لأن الطريقة المشتركة
' SharedFun()
' تستخدم عضو غير مشترك
Class My_Class
    Public X As Integer

    Public Shared Function SharedFun() As Integer
        Return X
    End Function
End Class
```

يمكنك الاستفادة من فكرة الطرق المشتركة ان كنت تطور مكتبة فئات وتود توفير بعض الطرق لمستخدمي الفئات دون الحاجة لتعريف كائنات جديدة، ولا بد أنك لاحظت أننا طيلة هذه الفترة استخدمنا الفئة ArabicConsole واستدعينا الطريقة WriteLine() دون الحاجة لإنشاء نسخة كائن جديدة، أي ان كل الذي فعله مؤلف الكتاب هو جعل الطريقة WriteLine() مشتركة، وبذلك سمحت لك باستخدامها مباشرة:

```
ArabicConsole.WriteLine("طريقة مشتركة")
```

إن كانت جميع أعضاء الفئة مشتركة، فيمكنك اعتبار الفئة كالوحدة البرمجية Module حيث ان الوحدات البرمجية ما هي إلا فئات جميع أعضائها مشتركة، ولكن نقطة الاختلاف الرئيسية بينها وبين الفئات هي أن الفئات يمكننا إنشاء نسخ منها تتمثل في كائنات. أخيراً، إن عرفت الإجراء Sub Main كوظيفة مشتركة داخل الفئة، يمكنك استدعائها مع بداية تنفيذ البرنامج من صندوق الحوار Project Property Pages (شكل 2-1 صفحة 32).

الخصائص المشتركة Shared Properties

لا توجد أي إضافات أخبرك بها هنا عن الخصائص المشتركة فهي بالضبط مثل الطرق المشتركة، أي يمكن استدعائها دون الحاجة إلى تعريف كائن ولا يمكن ان تستخدم الشيفرة التي بداخلها أعضاء غير مشتركة، فسأكتفي بعرض هذا المثال لتعريف خاصية مشتركة:

```

Shared Property SharedProp() As Integer
    Get
        ...
    End Get

    Set (ByVal Value As Integer)
        ...
    End Set
End Property

```

الأحداث المشتركة Shared Events

أيضا، الأحداث المشتركة ما هي إلا أحداث تتأثر بها جميع الكائنات المنشئة من فئة لحظة انطلاقها، وحتى أوضح لك المنطق البرمجي لها، دعني اضرب مثال واقعي: عندما يموت احد الأشخاص فان الموت حدث يقع على ذلك الشخص لوحده، وردة الفعل أو تأثير الحدث سيكون على الشخص وحده فقط، أما إن وقعت كارثة جماعية (كوقوع زلزال) فان جميع الكائنات ستتأثر بهذا الحدث ويلقوا حتفهم. لتعرف حدث مشترك استخدم الكلمة المحجوزة Shared أيضا:

```

Class PersonClass
    Event Die()
    Shared Event AllDie() ' حدث مشترك

    Sub KillHim()
        RaiseEvent Die()
    End Sub

    Shared Sub EarthQuick()
        RaiseEvent AllDie()
    End Sub
End Class

```

ستلاحظ أن الطريقة المشتركة EarthQuick() هي التي ستفجر حدث الزلزال (لا يشترط أن تكون الطريقة مشتركة حتى يتم إطلاق حدث مشترك)، والنقطة الرئيسية التي عليك معرفتها هو أن حدث الزلزال AllDie سيتم إطلاقه في كافة الكائنات المنشئة من نفس الفئة، دعنا نرى تأثير الحدث Die العادي أولا:

```

Module Module1
    Sub Main()
        Dim Turki As New PersonClass()
        Dim Ali As New PersonClass()

        ' قمص الحدث لكلا الكائنين
        AddHandler Turki.Die, AddressOf PersonDie
        AddHandler Ali.Die, AddressOf PersonDie
    End Sub
End Module

```

```
Turki.KillHim()

Ali.KillHim()
End Sub

Sub PersonDie()
    ArabicConsole.WriteLine("توفي شخص")
End Sub
End Module
```

ستلاحظ أن عملية إطلاق الحدث Die مرتبطة بكل كائن بشكل مستقل، فالإجراء PersonDie() السابق سيتم استدعائه مرتين بسبب استخدام الطريقة KillHim() مع كل كائن على حده، أما لو جربنا إطلاق الحدث المشترك AllDie فلن نحتاج إلا إلى عملية قنص واحدة فقط، وسيتم استدعائها لحظة إطلاق الحدث بغض النظر عن الكائن التابع له:

```
Module Module1
    Sub Main()
        Dim Turki As New PersonClass()
        Dim Ali As New PersonClass()

        ' قنص واحد فقط للحدث المشترك
        AddHandler PersonClass.AllDie, AddressOf PersonDie

        Turki.EarthQuick()
        Ali.EarthQuick()

        ' طريقة مشتركة لذلك يمكن
        ' استدعائها مباشرة
        PersonClass.EarthQuick()
    End Sub

    Sub PersonDie()
        ArabicConsole.WriteLine("توفي شخص")
    End Sub
End Module
```

تفدك فكرة الأحداث المشتركة كثيرا إذا أردت قنص حدث واحد فقط لمجموعة كائنات مختلفة، فعندما نصل إلى الجزء الثالث تطوير تطبيقات Windows من هذا الكتاب، سننشئ مجموعة من الأدوات بحيث نتشارك جميعها في حدث واحد.

الصيغة القياسية للأحداث في عالم NET :

ان كانت لي معزة صغيرة في قلبك أتمنى من هذه اللحظة حتى نهاية حياتك البرمجية مع Visual Basic استخدام صيغة عالم NET. القياسية لتعريف الأحداث (سواء كانت مشتركة أو عادية)، فكما لاحظت سابقا أن كل الإجراءات يمكن أن تقنص الأحداث وتكون قابلة للتنفيذ لحظة انطلاق الحدث، وكما علمت أيضا أن جميع الإجراءات يمكنها أن تستقبل أحداث الكائنات المختلفة، لذلك اعتمد مبرمجو NET على توحيد صيغة تعريف الأحداث حتى تكون قابلة للعمل على جميع الإجراءات.

ليس هذا فقط، بل حتى عندما نتوغل في عالم برمجة إطار عمل NET Framework. ستلاحظ ان جميع الأحداث في جميع الكائنات المختلفة صيغتها موحدة. صحيح ان تطبيق هذه الصيغة قد يكلفك وقت ومجهود إضافي، إلا انك ستوفر الكثير من هذا المجهود لاحقا، وبالذات ان كبر حجم برامبك ومشاريعك.

القاعدة الأولى التي عليك معرفتها لتطبيق الصيغة القياسية هي: جميع الأحداث التي تعرفها ترسل لها وسيطتين الأولى تسمى sender والثانية e:

```
Class PersonClass
    Event Die(ByVal sender As Object, ByVal e As System.EventArgs)
    ...
End Class
```

حيث sender هو الكائن الذي أطلق فيه الحدث، بينما e تمثل جميع الوسيطات Parameters التي يرسلها الحدث، وبما ان الحدث Die السابق لن يقوم بإرسال أية وسيطات إضافية، لذلك استخدمنا الفئة System.EventArgs والمقدمة من إطار عمل NET Framework. حيث لن تحتوي على اية وسيطات، لذلك عند إطلاق الحدث أنشئ الكائن عند إرساله للوسيلة مباشرة:

```
Class PersonClass
    ...
    Sub KillHim()
        RaiseEvent Die(Me, New System.EventArgs())
    End Sub
End Class
```

والآن يمكنك قنص الحدث بنفس الطريقة إما باستخدام WithEvents أو AddHandler:


```
Module Module1
    Sub Main()
        Dim Turki As New PersonClass()
        AddHandler Turki.Die, AddressOf PersonHasDied

        Turki.KillHim()
    End Sub

    Sub PersonHasDied(ByVal sender As Object, ByVal e As _
        System.EventArgs)

        ArabicConsole.WriteLine("لقد توفي الشخص")
    End Sub
End Module
```


من ناحية أخرى، إذا أردت من الحدث إرسال وسيطات إضافية وكنت تود الاستمرار على صيغة .NET. القياسية للأحداث، عليك تعريف فئة جديدة خاصة لوسيطات الحدث (مع ضرورة اشتقاق الفئة `System.EventArgs` وراثيًا باستخدام الكلمة المحجوزة `Inherits`)، مثلًا حدث السفر `Travel`:

```

Class TravelEventArgs
    Inherits System.EventArgs

    Public FromCity As String
    Public ToCity As String

    Sub New(ByVal fromCity As String, ByVal toCity As String)
        Me.FromCity = fromCity
        Me.ToCity = toCity
    End Sub
End Class
```

والآن سنعود إلى الفئة `PersonClass` مرة أخرى لنعرف فيها الحدث، وطريقة تمكنا من إطلاقه:

```

Class PersonClass
    Event Travel(ByVal sender As Object, ByVal e As TravelEventArgs)

    Sub Move(ByVal fromCity As String, ByVal toCity As String)
        RaiseEvent Travel(Me, New TravelEventArgs(fromCity, toCity))
    End Sub
    ...
End Class
```

أخيراً، الشيفرة المسطورة في الصفحة التالية توضح لك طريقة قنص الحدث Travel السابق وهي شبيهة تماماً بالطرق السابقة، ولا تنسى تعديل الوسيطات حتى تتم عملية القنص بشكل صحيح، يمكنك أيضاً استخدام WithEvents أو AddHandler):

```
Module Module1
    Sub Main()
        Dim Turki As New PersonClass()
        AddHandler Turki.Travel, AddressOf PersonHasTraveld

        Turki.Move("الرياض", "الطهران")
    End Sub

    Sub PersonHasTraveld(ByVal sender As Object, ByVal e As
TravelEventArgs)

        ArabicConsole.WriteLine("سافر الشخص من" & e.FromCity _
& " إلى " & e.ToCity)

    End Sub
End Module
```

بعد هذا الفصل الطويل جداً والمعقد جداً عرفتكم على أهم مواضيع برمجة Visual Basic .NET والذي يتعلق بالفئات Classes والكائنات Objects وكيفية التعامل معها. لا تحاول الانتقال إلى الفصل التالي حتى تتأكد من مدى استيعابك لهذا الفصل، حيث أن جميع فصول الكتاب وبرمجة إطار عمل .NET Framework بشكل عام تعتمد اعتماد كلي على الفئات والكائنات. إن كنت مستوعباً لمحتويات هذا الفصل، فمرحباً ألف بك مع عالم الوراثة عنوان الفصل التالي.

الوراثة

لا أستطيع إعطائك نسبة معينة تمثل أهمية موضوع الوراثة في Visual Basic .NET، ولكنني سأخبرك ببساطة ان جميع فئات مكتبة إطار عمل NET Framework متوارثة فيما بينها. وان لم تكن جادا في استيعاب هذا المبدأ، فستصادف الكثير من المتاعب عندما تتوغل في استخدام مكتبة فئات إطار عمل NET Framework عاجلا أو آجلا.

هذا الفصل هو مدخلك الرئيسي للوراثة ومواضيع أخرى لها صلة، ولا أنزع سرا إن أخبرتك أنني استمتع جدا في الحديث عن هذا الموضوع، لذلك سأبدأ معك من الصفر وسأفترض انك لا تعلم أي شيء عن هذا المبدأ، وأعدك أنني سأحاول جعل مادة هذا الفصل خفيفة وممتعة وقابلة للفهم السريع بمشيئة الله، وكل ما اطلبه منك هو التركيز في الشيفرات المصدرية، فهي مفتاح المعرفة لاستيعاب مبدأ الوراثة.

مقدمة إلى الوراثة

هذا القسم من الفصل هو مدخلك المبدئي إلى الوراثة وكيفية تطبيقها بـ Visual Basic .NET.

مبدأ الوراثة

إن كنت على دراية كافية بمبدأ الوراثة، يمكنك الانتقال إلى فقرة تطبيق الوراثة بـ Visual Basic .NET دون أي مشاكل، أما إن كان هذا المصطلح جديدا عليك، فأستطيع أن اعرف لك الوراثة Inheritance على أنها قدرة الفئة (الفئة المشتقة Derived Class) على اشتقاق أعضاء من فئة أخرى (الفئة القاعدية Base Class) دون الحاجة لإعادة تعريفها من جديد. فلو كانت لدينا فئة تمثل شخص Person بها خصائص كالاسم Name والعمر Age، يمكننا اشتقاق فئة أخرى منها Employee لتراث الخاصيتين من الفئة القاعدية (Name و Age) بالإضافة إلى بعض الأعضاء الخاصة بها، لنستطيع كتابة شيئا مثل:

```
Dim Turki As New Employee

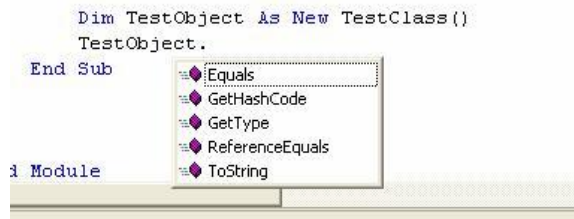
' أعضاء من الفئة القاعدية
' Person
Turki.Name = "تركى العسيري"
Turki.Age = 99

' أعضاء إضافية من الفئة
' Employee
Turki.Job = "مطور برامج ومواقع ويب"
Turki.Salary = 100
...
```

المزايا التي تجنيها من الوراثة لا تعد ولا تحصى، فيكفي انك تستطيع تطوير فئة معينة وذلك باشتقاقها وراثيا لتعريف فئة أخرى. من ناحية أخرى، ان اكتشفت احد الأخطاء في تصميم الفئة القاعدية (الفئة Person مثلا)، فلست بحاجة إلى تنقيح الفئة المشتقة من جديد (Employee)، حيث ان الفئة المشتقة ستتأثر تلقائيا بجميع التعديلات التي تجريها على الفئة القاعدية. المزيد أيضا، لست بحاجة إلى إعادة تكرار عملية بناء الفئات للأعضاء القياسية لتوفير الوقت، فعندما تنشئ الفئة Employee لن تضطر إلى إعادة تعريف الخصائص Age و Name فانك ستستقها من الفئة Person، وكفي إضافة الأعضاء التي تود بنائها داخل الفئة المشتقة فقط.

جميع الفئات المعرفة في مكتبة فئات إطار عمل NET Framework. تستخدم الوراثة، والدليل على ذلك لو أنك عرفت فئة خالية وأنشئت كائن منها، ستلاحظ وجود طرق إضافية (شكل 1-4):

```
Class TestClass
    ' فئة لا تحتوي على أية أعضاء
End Class
```



شكل 1-4: أعضاء إضافية تابعة للكائن رغم أن الفئة لا تحتوي على أية أعضاء.

من أين أتت هذه الأعضاء الإضافية؟ والجواب بكل بساطة من الفئة System.Object والتي ترث منها جميع مكتبة فئات إطار عمل .NET Framework. الأخرى -كما ستري لاحقاً في الفصل السادس الفئات الأساسية.

لغويًا، تسمى العلاقة -في لغات OOP- بين الفئات المشتقة بعلاقة **هو Is a**، فعند الحديث عن فئة الموظف Employee يمكننا ان نطلق عليه عبارة **هو شخص** Person.

تطبيق الوراثة بـ Visual Basic .NET

والآن دعني اعرض عليك مثالا مبسطا يمكنك من استيعاب الوراثة بشكل تطبيقي، عرف هذه الفئة التي تمثل شخص في احد ملفات المشروع:



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

كل ما عليك فعله لتطبيق مبدأ الوراثة بـ Visual Basic .NET هو استخدام الكلمة المحبوزة Inherits وإلحاقها باسم الفئة المراد اشتقاقها وراثيًا:



```
Class Employee
    Inherits Person ' Person فئة الوراثة أعضاء الفئة
    Public Job As String
    Public Salary As Double
End Class
```

يمكنك البدء فوراً الآن بإنشاء كائن من الفئة المشتقة ليصل إلى جميع أعضاء الفئة القاعدية والفئة المشتقة:



```
Sub Main()
    Dim Turki As New Employee()

    ' أعضاء الفئة القاعدية
    Turki.Name = "تركي العسيري"
    ...

    ' أعضاء الفئة المشتقة
    Turki.Job = "مطور برامج ومواقع ويب"
    ...
End Sub
```

```

        ArabicConsole.WriteLine(Turki.Name) ' تركي العسيري
        ArabicConsole.WriteLine(Turki.Job)   ' مطور برامج ومواقع ويب
    ...
End Sub

```

ضع في اعتبارك أن الفئة Employee هي الوارثة من الفئة Person والعكس غير صحيح، فلو حاولت تعريف كائن من الفئة القاعدية، لن تتمكن من الوصول إلى الفئة المشتقة (لان الفئة Person لا تعلم انه تم اشتقاقها):

```

Dim Turki As New Person()

' ممكن كما علمنا
Turki.Name = "تركي العسيري"

' مستحيل
Turki.Job = "مطور برامج ومواقع ويب"

```

من ناحية أخرى، تستطيع اشتقاق الفئة Person أكثر من مرة لتعرف فئة أخرى بنفس الطريقة:

```

Class Student
    Inherits Person

    Public Department As String
    Public Grade As Integer
End Class

```

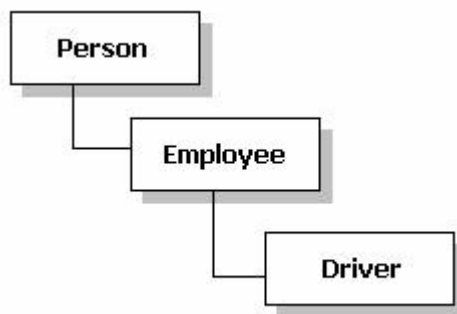
بل يمكنك أيضا اشتقاق فئة مشتقة أخرى، فيمكنك مثلا تعريف الفئة Driver المشتقة من الفئة Employee المشتقة من الفئة Person (شكل 4-2 بالصفحة المقابلة):

```

Class Driver
    Inherits Employee

    Public CarModel As String
    Public LicenseNumber As String
End Class

```



شكل 4-2: شكل يوضح العلاقة الوراثية بين الفئات.

وبشكل منطقي، الكائنات المنشئة من الفئة Driver يمكنها الوصول إلى أعضاء الفئة Employee والفئة Person:

```

Dim Abbas As New Driver()

' أعضاء من Person
Abbas.Name = "عباس السريع"
...

' أعضاء من Employee
Abbas.Job = "سائق خاص"
...

' أعضاء من Driver
Abbas.CarModel = "BMW - 7 Class"
...
    
```

ملاحظة

إن كنت من مبرمجي OOP المخضرمين، دعني أوضح لك أن (شكل 4-2) يبين العلاقة الوراثية بين الفئات وليس علاقة الاحتواء Containment بين الكائنات.

مع ذلك، لا يمكن للفئة الواحدة أن تشق أعضاء من أكثر من فئة وتطبيق ما يسمى الوراثة المتعددة **Multiple Inheritance** (والذي لا تدعمه كل لغات .NET. الأخرى باستثناء Visual C++):

```

Class Thing
    Inherits Person
    Inherits Animal ' رسالة خطأ '
    ...
End Class

```

التعامل مع الفئات الوراثة والمورثة

في الصفحات السابقة عرفتكم على الفكرة الأساسية من الوراثة والفئات القاعدية Base Classes والفئات المشتقة Derived Classes، ووضحت لك كيف تطبق مبدأ الوراثة — Visual Basic .NET عن طريق استخدام العبارة Inherits، مع ذلك عليك معرفة الكثير من المسائل والحالات الخاصة التي ستواجهها عند تطبيق الوراثة على الفئات. الفقرات التالية ستخبرك بهذه التفاصيل.

وراثة الأعضاء

لا تقتصر عملية الاشتقاق الوراثي على حقول الفئات فقط، بل جميع الأعضاء الأخرى (الأحداث، الخصائص، والطرق) يتم اشتقاقها أيضاً:

```

Class Person
    ' حدث
    Event Die()

    ' خاصية
    Private m_BirthDate As Date
    Property BirthDate() As Date
        Get
            Return m_BirthDate
        End Get
        Set(ByVal Value As Date)
            m_BirthDate = Value
        End Set
    End Property

    ' طريقة
    Sub KillHim()
        RaiseEvent Die()
    End Sub
    ...
End Class

```

سيتم اشتقاق هذه الأعضاء في الفئة بمجرد استخدام العبارة Inherits، لنتمكن من الوصول إليها واستخدامها بنفس الأساليب السابقة:

```
Module Module1
    Sub Main()
        Dim Turki As New Employee()

        ' الحدث
        AddHandler Turki.Die, AddressOf PersonHasDied

        ' الخاصية
        Turki.BirthDate = #1/1/9999#

        ' الطريقة
        Turki.KillHim()
    End Sub

    Sub PersonHasDied()
        ArabicConsole.WriteLine ("لقد توفي الشخص")
    End Sub
End Module
```

ليس هذا فقط، بل حتى الأعضاء المشتركة Shared Members يتم وراثتها بنفس الطريقة، ولكن ضع في عين الاعتبار أن الأعضاء المشتركة شاملة لكل الفئات الوارثة والموروثة، فلو عرفت هذا الحقل المشترك:

```
Class Person
    Public Shared LastName As String
End Class
```

عليك معرفة أن هذا الحقل سيتبع الفئة القاعدية وجميع الفئات المشتقة، فلا تتوقع وجود نسختين منه (نسخة لـ Person وأخرى لـ Employee)، وكإثبات لكلامي أسوق لك هذا المثال:

```
Dim Turki As New Person()           ' كائن من الفئة القاعدية
Dim Abdullah As New Employee()       ' كائن من الفئة المشتقة

' تعديل قيمة الحقل المشترك للفئة القاعدية
Turki.LastName = "العسيري"

' هو نفسه الحقل الموجود في الفئة المشتقة
ArabicConsole.WriteLine(Abdullah.LastName) ' العسيري
```

الكلام السابق يطبق أيضا على الطرق المشتركة، الخصائص المشتركة، والأحداث المشتركة.

على صعيد آخر، يمكن للفئة المشتقة الوصول إلى جميع أعضاء الفئة القاعدية، شريطة ان تكون الأعضاء عرفت على مستوى Public أو Friend، أما الأعضاء الخاصة Private فلا يمكن الوصول لها إلا من داخل الفئة القاعدية نفسها:

```
Class Employee
    Inherits Person

    ...
    ...
    Sub MethodInEmployee()
        ' الوصول إلى أعضاء الفئة القاعدية
        Me.Name = "تركى العسيري"
        Me.Age = 99
    End Sub
End Class
```

مع ذلك، لن تستطيع الوصول إلى أعضاء الفئة القاعدية بالطريقة السابقة عند تشابه أسماء أعضاء الفئة القاعدية والفئة المشتقة، خاصة عند تطبيق ما يسمى إعادة القيادة Overriding كما سترى لاحقاً في القسم إعادة القيادة Overriding من هذا الفصل.

المشيدات Constructors

عند الحديث عن المشيدات، انسى كل ما ذكرته في الفقرة السابقة، وذلك لأن المشيدات لا يتم اشتقاقها بين الفئات كباقي الأعضاء لحظة الوراثة، ولكنها تتطلب منك كتابة شيفرات إضافية لتفعيلها مع الفئات القاعدية والمشتقة. وحتى اسهل عليك الأمر، دعنا نأخذ ثلاث حالات: الحالة الأولى عند وجود مشيد في الفئة المشتقة فقط دون الفئة القاعدية:

```
Class Person
    ...
    ' لا يوجد مشيد
    ...
End Class

Class Employee
    Inherits Person

    ...
    ...
    ' مشيد في الفئة المشتقة
    Sub New(ByVal Job As String)
        Me.Job = Job
    End Sub
End Class
```

في هذه الحالة، الغي كلمة الوراثة من خيالك وتعامل مع الفئة Employee وكأنها مستقلة ولم ترث أي شيء لحظة تمرير الوسيطات إلى المشيد:

```
Sub Main()  
    ' وسيطات مشيد الفئة المشتقة  
    Dim Turki As New Employee("مطور برامج ومواقع ويب")  
  
    Turki.Name = "تركي العسيري"  
    ArabicConsole.WriteLine(Turki.Job)  
    ...  
    ...  
End Sub
```

الحالة الثانية تكون عند وجود مشيد في الفئة القاعدية فقط دون الفئة المشتقة، عليك هنا بذل المستحيل وعمل كل ما تستطيع عمله حتى تتمكن من تنفيذ مشيد الفئة القاعدية باستخدام الكائن MyBase وليس Me (سأتحدث عن MyBase لاحقاً). وحتى تتمكن من عمل ذلك، عليك تعريف مشيد في الفئة المشتقة حتى لو لم تكن هناك حاجة إليه:

```
Class Person  
    ...  
    ' مشيد الفئة القاعدية  
    Sub New(ByVal Name As String)  
        Me.Name = Name  
    End Sub  
End Class  
  
Class Employee  
    Inherits Person  
  
    ...  
    ...  
    ' علينا تعريف هذا المشيد حتى نتتمكن  
    ' من استدعاء مشيد الفئة القاعدية  
    Sub New()  
        MyBase.New("تركي العسيري")  
    End Sub  
End Class
```

إن لم تطبق ما أخبرتك به في هذه الحالة، سيظهر لك مترجم .NET Visual Basic رسالة خطأ ولن تتمكن من تنفيذ البرنامج. أخيراً، عند إنشاء الكائن من الفئة Employee فلست بحاجة إلى إرسال أي وسيطات لمشيدها:

```

Sub Main()
    ' مشيد الفئة المشتقة لا يتطلب أي وسيطات
    Dim Turki As New Employee()

    Turki.Job = "مطور برامج ومواقع ويب"
    ArabicConsole.WriteLine(Turki.Name)
    ...
    ...
End Sub

```

الحالة الثالثة والأخيرة التي أود إخبارك بها تكون عند وجود مشيدات في كلا الفئتين (الفئة القاعدية والفئة المشتقة)، عليك إرسال وسيطات مشيد الفئة القاعدية من داخل الفئة المشتقة، وإرسال وسيطات مشيد الفئة المشتقة لحظة إنشاء الكائن:

```

Class Person
    ...
    ...
    Sub New(ByVal Name As String)
        Me.Name = Name
    End Sub
End Class

Class Employee
    Inherits Person

    ...
    ...
    Sub New(ByVal Name As String, ByVal Job As String)
        ' استدعي مشيد الفئة القاعدية أولا
        MyBase.New(Name)

        ' ثم قم بعمل ما تريد
        Me.Job = Job
    End Sub
End Class

Module Module1
    Sub Main ()
        ' لاستخدام الكائن ارسل وسيطات
        ' مشيد الفئة المشتقة
        Dim Turki As New Employee("تركبي العسيري", "مطور برامج ومواقع ويب")
        ...
        ...
    End Sub
End Module

```

قبل ان اختتم فقرة المشيدات، علي التنويه بضرورة إرسال القيم لوسيطات مشيد الفئة القاعدية باستخدام MyBase قبل كتابة أي حرف من شيفرات مشيد الفئة المشتقة، فالمشيد التالي سيظهر رسالة خطأ:

```
Sub New(ByVal Name As String, ByVal Job As String)
    ' رسالة خطأ لاننا لم نرسل القيم لوسيطات
    ' مشيد الفئة القاعدية اولا
    Me.Job = Job
    MyBase.New(Name)
End Sub
```

التعامل مع الكائنات

لننتقل من مرحلة تأليف وبناء الفئات إلى مرحلة استخدامها وإنشاء الكائنات، أنت تعلم وأنا اعلم أن المتغيرات من النوع Object يمكن إسناد أي قيمة لها:

```
Dim Turki As Object

Turki = New Employee()
...
...
```

إذا أردت معرفة السبب الذي يسمح لنا بفعل ذلك، يمكنني أن أخصه لك بعبارة: جميع أنواع البيانات -بما فيها الفئات- في عالم NET. مشتقة من النوع Object (الاسم الكامل له هو System.Object)، ومن الثمار الابتدائية التي تجنيها من الوراثة هو قدرتك على إسناد كائن من فئة مشتقة إلى كائن من فئة قاعدية، فالعملية التالية:

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

Turki.Name = "تركي العسيري"

Turki2 = Turki ' ممكن جدا

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

صحيحة مئة مية يا باشا، والسبب ان جميع أعضاء الفئة القاعدية Person موجودة ايضا في الفئة المشتقة Employee، لذلك سمح لنا NET. Visual Basic من إسناد قيمة الكائن Turki إلى الكائن Turki2. مع ذلك، لا يزال الكائن Turki2 منشأ من الفئة القاعدية فقط، فلا تحاول الوصول إلى أعضاء مشتقة في الفئة Employee -كما تفعل مع الكائن Turki:

```
' رسالة خطأ
Turki2.Salary = 1
```

يفضل قدرة إسناد قيم مشتقة إلى كائنات منشئة من فئات قاعدية، يمكنك -مثلاً- تعريف إجراء واحد فقط ليستقبل وسيطات من كائنات لفئات مشتقة مختلفة:

```
Sub PrintInfo(ByVal personObject As Person)
    ArabicConsole.WriteLine(personObject.Name)
    ArabicConsole.WriteLine(personObject.Age)
End Sub
```

لنتمكن من إرسال كائنات من النوع Person أو من جميع الأنواع الأخرى بشرط ان تكون مشتقة من Person:

```
Dim Turki As New Person()
Dim Ali As New Employee()
Dim Umar As New Student() ' مشتق من Person ايضا
...
...

' كل هذه الاستدعاءات صحيحة
PrintInfo(Turki)
PrintInfo(Ali)
PrintInfo(Umar)
```

ولمزيد من التوضيح، يسمح لك Visual Basic .NET بإسناد هذا النوع من القيم من كائن فئة مشتقة إلى كائن فئة أم فقط، فإن حاولت عمل العكس ستظهر رسالة خطأ:

```
Dim Abood As New Person()
Dim Abood2 As Employee

Abood.Name = "عبود اللوح"

Abood2 = Abood ' رسالة خطأ
```

مع ذلك، يمكنك الالتفاف حول رسالة الخطأ عن طريق انشاء نسخة جديدة من كائن الفئة Employee وإسناد الكائن إلى متغير من النوع Person:

```
' بافتراض ان
' Option Strict Off
Dim Abood As Person
Dim Abood2 As Employee
```

```

Abood = New Employee()
Abood.Name = "عبود اللوح"

Abood2 = Abood ' ممكن

ArabicConsole.WriteLine(Abood2.Name) ' عبود اللوح

```

كما هو موضح في تعليق الشيفرة السابق، لابد من تعطيل العبارة Option Strict Off حتى تتمكن من إسناد القيمة، اما ان كنت مبرمج لبق جدا وتفضل Option Strict On دائما في مشاريعك (كما هو الحال مع شيفرات هذا الكتاب)، فعليك استخدام المعامل CType:

```

' في حالة تفعيل
' Option Strict On
...
Abood2 = CType(Abood, Employee)
...

```

انظر أيضا

لمزيد من التفاصيل حول المعامل CType، راجع الفصل الثاني لغة البرمجة.

إن كنت من مبرمجي OOP المخضرمين، فقد تطلق على العمليات السابقة التعبير **تعدد الواجهات** **Polymorphism**، وفي الحقيقة تعبيرك في محله ولكن ينقصه شيء بسيط حتى يقبل على هذا التعبير، سأريك النقص في الفصل القادم **الواجهات**، **التفويض** و**المواصفات** بمشيئة الله.

إعادة القيادة Overriding

افتراض أننا نريد تعريف طريقة ShowName() في فئة Person لتعرض لنا الاسم الأول والأخير للشخص، ومن ثم نقوم باشتقاقها في فئة Employee:

```

Class Person
    Public FirstName As String = "عباس"
    Public LastName As String = "السريع"

    Sub ShowName()
        ArabicConsole.WriteLine(Me.FirstName & " " & Me.LastName)
    End Sub
End Class

```

```

Class Employee
    Inherits Person
    ...
    ...
End Class

```

وبعد ان قمت باشتقاق هذه الفئة لتعرف فئة الموظف Employee اكتشفت لاحقا أن الشركة التي تستخدم برنامجك لا تعتمد على هذه الصيغة في كتابة الأسماء، حيث أنها تبدأ باسم القبيلة أو العائلة أولاً ومن ثم الاسم الأول للشخص، قد تقوم بإعادة كتابة الطريقة ShowName() في الفئة القاعدية Person:

```

Class Person
    ...
    ...
    Sub ShowName()
        ArabicConsole.WriteLine(Me.LastName & ", " & Me.FirstName)
    End Sub
End Class

```

لقد اقترفت إما برمجياً وبهتانا عظيماً في التعديل السابق! والسبب انه قد توجد مئات الفئات الأخرى في البرنامج تشق الفئة السابقة Person لتؤثر على طرق تنسيقها بشكل سلمي، وما هي فائدة الوراثة إذاً إن قمت بتعديل الفئة القاعدية وأنت تريد تطوير فئة منها (الفئة المشتقة) دون المساس بالفئة القاعدية؟

أفضل حل سنقوم بتطبيقه يتم بإعادة كتابة الطريقة ShowName() في الفئة المشتقة Employee بحيث تكون هذه الطريقة خاصة بها ولن تتأثر لا الفئة القاعدية ولا الفئات الأخرى المشتقة منها بهذه الإضافة، وهذا بالضبط ما يسمى **إعادة القيادة Overriding**. حتى تسمح للطريقة المعرفة في الفئة القاعدية من إعادة قيادتها، عليك استخدام الكلمة المحجوزة :Overridable

```

Class Person
    Public FirstName As String = "عباس"
    Public LastName As String = "السرير"

    ' لنسمح بإعادة قيادة هذه الطريقة
    ' في الفئات المشتقة من هذه الفئة
    Overridable Sub ShowName()
        ArabicConsole.WriteLine(Me.FirstName & " " & Me.LastName)
    End Sub
End Class

```

والآن يمكنك إعادة قيادة هذه الطريقة متى ما شئت لحظة الاشتقاق الوراثي بالحقاق الكلمة المحجوزة Overrides قبل اسم الطريقة:

```
Class Employee
    Inherits Person

    ' إعادة قيادة الطريقة
    Overrides Sub ShowName()
        ArabicConsole.WriteLine(Me.LastName & ", " & Me.FirstName)
    End Sub
End Class
```

عند استدعاء هذه الطريقة لكائن من نوع Person سيتم تنفيذ الطريقة التابعة للفئة القاعدية، وعند استدعائها لكائن من نوع Employee سيتم تنفيذ الطريقة المعاد قيادتها:

```
Dim Abbas As New Person()
Dim Abbas2 As New Employee()

Abbas.ShowName() ' عباس السريع
Abbas2.ShowName() ' السريع، عباس
```

إعادة قيادة الطرق والخصائص

عملية إعادة القيادة تقتضي إعادة بناء وكتابة الإجراء كي يناسب الفئة المشتقة، فذلك من البديهي أنك لن تستطيع إعادة قيادة إلا الطرق Methods والخصائص Properties فقط. بالنسبة للطرق، فقد عرضت في الفقرة السابقة مثالاً لإعادة قيادتها، ولن تتمكن من فعل ذلك إلا عند استخدام الكلمة المحجوزة Overridable قبل اسم الطريقة في الفئة القاعدية، وفي كل مرة نود إعادة قيادة طريقة في فئة مشتقة استخدم الكلمة المحجوزة Overrides.

استخدامك للكلمة Overrides في الفئة المشتقة شبيه باستخدام Overridable في الفئة القاعدية من ناحية صلاحية إعادة القيادة، والدليل أنك لو حاولت اشتقاق الفئة المشتقة Employee لتعرف فئة جديدة (Driver مثلاً) فستتمكن أيضاً من إعادة قيادة طرقها والمعرفة بـ :Overrides

```
Class Driver
    Inherits Employee

    Overrides Sub ShowName()
        ' يمكنك إعادة قيادة الطريقة للمرة الثالثة
        ...
    End Sub
End Class
```

نستنتج من كل ذلك، ان Overrides تؤدي عمل Overridable ولكن Overridable لا تؤدي عمل Overrides. ففي المثال السابق تمكنا من إعادة قيادة الطريقة ShowName() من الفئة Employee، وحتى تمنع حدوث ذلك يمكنك استخدام الكلمة المحجوزة NotOverridable:

```
Class Employee
    Inherits Person

    ' منع الفئات المشتقة من هذه الفئة
    ' من إعادة قيادة هذه الطريقة
    NotOverridable Overrides Sub ShowName()
        ArabicConsole.WriteLine(Me.LastName & ", " & Me.FirstName)
    End Sub
End Class
```

ملاحظة

ذكرت قبل قليل أن مبدأ إعادة القيادة قابل للتطبيق على الطرق والخصائص فقط، وهذا لا يشمل الطرق والخصائص المشتركة حيث أنك لن تستطيع إعادة قيادتها.

بالنسبة للخصائص، فجميع ما ذكرته حول الطرق سابقا يطبق عليها تماما، مع العلم أنك لن تتمكن من تغيير قابلية القراءة والكتابة عند إعادة قيادة الخاصية، فمثلا لو عرفت خاصية في الفئة القاعدية للقراءة فقط ReadOnly:

```
Class Person
    ...
    ' خاصية للقراءة فقط
    Overridable ReadOnly Property BirthDate() As Date
        Get
            ...
        End Get
    End Property
End Class
```

فلن نتمكن من جعلها للكتابة فقط WriteOnly لحظة إعادة قيادتها:

```
Class Employee
    Inherits Person
    ...
    ' رسالة خطأ في تعريف الخاصية
    Overrides WriteOnly Property BirthDate() As Date
        Set(ByVal Value As Date)
            ...
        End Set
    End Property
End Class
```

والعكس صحيح، فلن تتمكن من إعادة قيادتها وجعلها للقراءة فقط إن كانت معرفة للكتابة فقط في الفئة القاعدية، وحتى لو كانت قابلة للقراءة والكتابة، فلا بد من أن تكون أيضاً قابلة للقراءة والكتابة عند إعادة قيادتها. وبالنسبة للخصائص الافتراضية Default Properties، فلا بد من أن تكون الخاصية افتراضية عند إعادة قيادتها في الفئة المشتقة.

انظر أيضا

لمزيد من التفاصيل حول الخصائص الافتراضية Default Properties، راجع الفصل الثالث **الفئات والكائنات**.

نقطة أخيرة حول إعادة القيادة وهي ان استدعاء الإجراء المعاد قيادته يعتمد على نوع الكائن المنشأ وليس نوع المتغير المعلن (المؤشر):

```
Dim Abbas As Person = New Employee()
Dim Abbas2 As Employee = New Employee()
```

```
Abbas.ShowName() ' السريع، عباس
Abbas2.ShowName() ' السريع، عباس
```

فكما تلاحظ، رغم ان المؤشر Abbas السابق من النوع Person الا انه استدعى الطريقة المعاد قيادتها في الفئة المشتقة Employee.

حالة إعادة التعريف Overloading:

دائماً ضع في اعتبارك، أن الفئة المشتقة والفئة القاعدية كلاهما -تقريباً- فئة واحدة (من نظرة الفئة المشتقة)، فيمكنك مثلاً إعادة كتابة طريقة بإعادة تعريفها Overloads عوضاً عن إعادة قيادتها :Overrides

```
Class BaseClass
    Overloads Sub TestMethod()
        ArabicConsole.WriteLine("الفئة القاعدية")
    End Sub
End Class

Class DerivedClass
    Inherits BaseClass

    Overloads Sub TestMethod(ByVal x As Integer)
        ArabicConsole.WriteLine("الفئة المشتقة")
    End Sub
End Class
```

وكما ذكرت في الفصل السابق، يمكنك استخدام الكلمة المحجوزة Overloads لإعادة تعريف طريقة ما شريطة اختلاف نوع وسيطاتها مع الطرق الأخرى، ولكن عند الحديث عن الفئات المشتقة فالوضع يختلف قليلاً، قد تفاجئك هذه الشيفرة:

```
Class BaseClass
    Overloads Sub TestMethod()
        ArabicConsole.WriteLine("الفئة القاعدية")
    End Sub
End Class

Class DerivedClass
    Inherits BaseClass

    ' لاحظ عدم تغيير وسيطات الطريقة
    ' المعاد تعريفها
    Overloads Sub TestMethod()
        ArabicConsole.WriteLine("الفئة المشتقة")
    End Sub
End Class
```

كما نرى، سمح لنا مترجم Visual Basic .NET بإعادة تعريف الطريقة TestMethod() دون تغيير وسيطاتها! وحتى تتأكد من أن هذا ليس خطأ، يمكنك تعريف الكائنات وتجرب ذلك بنفسك:

```
Dim obj As New BaseClass()
Dim obj2 As New DerivedClass()

obj.TestMethod()      ' الفئة القاعدية
obj2.TestMethod()     ' الفئة المشتقة
```

قد تتساءل عن الفرق بين إعادة القيادة Overriding وإعادة التعريف Overloading في هذه الحالة، حيث أن الطريقتين تعملان بنفس الأسلوب في الحالتين، مع ذلك فالاختلاف بين استخدام Overrides واستخدام Overloads تقني من نظرة قابلية الوصول إلى الكائنات. حتى أوضح لك الفرق، أضف هذه الطرق إلى الفئات السابقة:

```
Class BaseClass
    ...
    ...
    Overridable Sub TestMethod2()
        ArabicConsole.WriteLine("الفئة القاعدية")
    End Sub
End Class

Class DerivedClass
    Inherits BaseClass
    ...
    ...
    Overrides Sub TestMethod2()
        ArabicConsole.WriteLine("الفئة المشتقة")
    End Sub
End Class
```

عند استدعاء الطرق من الكائنات، فإن الطرق المعاد قيادتها Overrides سيتم استدعاؤها اعتماداً على الكائن المنشأ وليس نوع المؤشر، أما استدعاء الطرق المعاد تعريفها Overloads فسيعتمد على نوع مؤشر الكائن وليس نوع الكائن:

```
Dim obj As BaseClass = New DerivedClass()
Dim obj2 As DerivedClass = New DerivedClass()

obj.TestMethod()      ' الفئة القاعدية
obj2.TestMethod()     ' الفئة المشتقة

obj.TestMethod2()     ' الفئة المشتقة
obj2.TestMethod2()    ' الفئة المشتقة
```

ملاحظة

استخدامك للكلمة المحجوزة Overloads عوضا عن Overrides كما
في المثال السابق، يؤدي إلى ما يسمى **بالتظليل** Shadowing كما
سترى بعد فقرتين.

استخدام MyBase

إن كان استخدام المؤشر Me يؤدي إلى الوصول إلى عضو في الكائن الحالي، فإن المؤشر
MyBase يؤدي إلى الوصول إلى أعضاء الفئة القاعدية. مبدئيا، عمل المؤشر Me يشابه عمل
المؤشر MyBase:

```
Class BaseClass
    Sub MethodInBase()
        ...
    End Sub
End Class

Class DerivedClass
    Inherits BaseClass

    Sub MethodInDerived()
        ' استدعاء الطريقة في الفئة القاعدية
        MyBase.MethodInBase()

        ' استدعاء الطريقة في الفئة القاعدية أيضا
        Me.MethodInBase()
    End Sub
End Class
```

استنادا إلى المثال السابق، صحيح أن Me يمكننا من استدعاء أعضاء الفئة القاعدية ولكن في حالة
عدم وجود طريقة معاد قيادتها Overriden، أما هنا:

```
Class BaseClass
    ...
    ...
    Overridable Sub MyMethod()
        ...
    End Sub
End Class
```

```
Class DerivedClass
    Inherits BaseClass
    ...
    ...
    Overrides Sub MyMethod()
        ...
    End Sub

    Sub TestMethod()
        ' استدعاء الطريقة في الفئة القاعدية
        MyBase.MyMethod()

        ' استدعاء الطريقة في الفئة المشتقة
        Me.MyMethod()
    End Sub
End Class
```

فيتضح لنا ان Me استدعت الطريقة -المعاد قيادتها- في الفئة الحالية، بينما MyBase موجهة للفئة القاعدية بشكل حصري.

فرق آخر بين MyBase و Me يتعلق بقدرة الوصول إلى الأعضاء الخاصة Private، حيث -كما أخبرتك في الفصل السابق- يمكنك Me من الوصول لها، بينما MyBase لا تسمح لك بذلك:

```
Class BaseClass
    Private fieldBase As Integer
End Class

Class DerivedClass
    Inherits BaseClass

    Private fieldDerived As Integer

    Sub Test()
        ' ممكن جدا
        Me.fieldDerived = 99

        ' رسالة خطأ
        MyBase.fieldBase = 99
    End Sub
End Class
```

بصفة عامة، الأعضاء الخاصة Private لا يمكن اشتقاقها من الفئة القاعدية.

الذي كنت أود أن أوصله لك من هذه الفقرة، هو ضرورة استخدام MyBase دائما إن قمت بإعادة تعريف المهدمات Finalize() أو Dispose() في الفئة المشتقة حتى تستدعي نفس المهدم التابع للفئة القاعدية:

```
Class DerivedClass
    Inherits BaseClass
    ...
    ...
    Protected Overrides Sub Finalize()
        ' استدعاء المهدم Finalize في الفئة القاعدية
        MyBase.Finalize()
    ...
End Sub
End Class
```

انظر أيضا

تحدثت عن المهدمات Finalize() و Dispose() في الفصل السابق **الفئات والكائنات**، وبالنسبة لمحدد الوصول المحمي Protected فساتطرق له لاحقا في هذا الفصل.

إن لم تقم باستدعاء المهدم MyBase.Finalize() فلن يتم تنفيذ مهدم الفئة القاعدية أبدا، لذلك احرص على استدعائه دائما إن قمت بإعادة قيادة المهدم Finalize فقط، أما إن لم تعيد قيادته فسيتم تنفيذ مهدم الفئة القاعدية تلقائيا -دون الحاجة لاستدعائه.

استخدام MyClass

من المعروف أن استخدام Me يمكنك من الوصول إلى أعضاء الفئة الحالية، ولكن عند إعادة قيادة العضو (أثناء عملية الوراثة بكل تأكيد)، فإن العضو المعاد قيادته هو الذي سيتم استدعائه وليس العضو التابع للفئة الحالية:

```
Class BaseClass

    Sub Test()
        ' رغم استخدام Me الا انه سيتم استدعاء
        ' طريقة الفئة المشتقة
        Me.OverridenMethod()
    End Sub
```

```

    Overridable Sub OverridenMethod()
        ArabicConsole.WriteLine("من الفئة القاعدية")
    End Sub
End Class

Class DerivedClass
    Inherits BaseClass

    Overrides Sub OverridenMethod()
        ArabicConsole.WriteLine("من الفئة المشتقة")
    End Sub
End Class

```

عند إنشاء نسخة كائن من الفئة المشتقة، ستلاحظ ان الطريقة المعاد قيادتها هي التي تم استدعائها من داخل الفئة القاعدية باستخدام Me:

```

Dim obj As New DerivedClass()
obj.Test() ' من الفئة المشتقة

```

من هنا يأتي دور الكلمة المحجوزة MyClass، حيث أنها تجبر مترجم اللغة على استدعاء العضو التابع للفئة الحالية ان تمت إعادة قيادته، اما استخدامها فهو مثل استخدام Me:

```

Class BaseClass
    Sub Test()
        ' هنا سيتم استدعاء طريقة الفئة الحالية
        MyClass.OverridenMethod()
    End Sub
    ...
    ...
End Class
...

```

التظليل Shadowing

قد تلجأ إلى اعتماد التظليل Shadowing في اغلب الأحوال، وبالأذات إن كانت قدرة الوصول إلى الشيفرة المصدرية للفئة القاعدية غير ممكنة (كوجود الفئة القاعدية في مكتبة DLL مثلاً)، فقد يكون مؤلف الفئة قد عرف طريقة دون استخدام الكلمة المحجوزة Overridable:

```

Class BaseClass
    Sub Method()
        ...
    End Sub
End Class

```

الطريقة السابقة لا يمكنك إعادة قيادتها (باستخدام Overrides) في الفئة المشتقة، والسبب ان مؤلف الفئة لم يسمح بذلك عن طريق الكلمة المحجوزة Overridable، مع ذلك يمكنك Visual Basic .NET من استخدام نفس اسم الطريقة في الفئة المشتقة (سيظهر المترجم تنبيه Warning لحظة الترجمة، ولكن الشيفرة سيتم تنفيذها دون مشاكل):

```
Class DerivedClass
    Inherits BaseClass

    Sub MyMethod()
        ...
    End Sub
End Class
```

لغويا نقول: ان الطريقة MyMethod() في الفئة المشتقة **ظللت Shadows** الطريقة MyMethod() في الفئة القاعدية، ومن نظرة كائنية، فسيتم تنفيذ الطريقة الموجودة في الفئة المشتقة:

```
Dim obj As New DerivedClass()
obj.MyMethod() ' استدعاء الطريقة في الفئة المشتقة
```

استخدم الكلمة المحجوزة Shadows في كل مرة تريد تظليل طريقة (لن تظهر الإنذارات Warnings في هذه الحالة):

```
Class DerivedClass
    Inherits BaseClass

    Shadows Sub MyMethod()
        ...
    End Sub
End Class
```

ملاحظة

رغم ان ظهور الإنذارات Warnings أثناء ترجمة البرنامج لا تؤدي إلى وقف التنفيذ، إلا أنني أنصحك بعنف بتعديل الشيفرات التي تسبب هذه الإنذارات دائما قبل توزيع البرنامج.

حالة إعادة التعريف Overloading مرة أخرى:

استخدام الكلمة المحجوزة Overloads يؤدي إلى تظليل الطريقة أيضا - كما تفعل الكلمة المحجوزة Shadows:

```
Class DerivedClass
    Inherits BaseClass

    ' Overloads باستخدام
    Overloads Sub MyMethod()
        ...
    End Sub
End Class
```

مع ذلك توجد فروق بين استخدام Shadows و Overloads عند التظليل، الفرق الأول هو أن Shadows يمكنك استخدامها مع كافة أنواع أعضاء الفئة، بينما Overloads موجه للطرق والخصائص فقط. الفرق الثاني - وهو الأهم - أن Shadows تقوم بتظليل كافة الطرق والخصائص المعاد تعريفها في الفئة القاعدية، بينما Overloads تظلل الطريقة أو الخاصية التي تحمل نفس بسيطاتها فقط، وحتى تستوعب ما اقصده، افترض هذه الفئة:

```
Class A
    Sub MyMethod()
        ArabicConsole.WriteLine("A.MyMethod")
    End Sub

    Sub MyMethod(ByVal x As Integer)
        ArabicConsole.WriteLine("A.MyMethod (x)")
    End Sub

    Sub MyMethod2()
        ArabicConsole.WriteLine("A.MyMethod2")
    End Sub

    Sub MyMethod2(ByVal x As Integer)
        ArabicConsole.WriteLine("A.MyMethod2 (x)")
    End Sub
End Class
```

والآن سأقوم باشتقاق الفئة وتظليل طريقتين منها - الأولى باستخدام Overloads والثانية بـ Shadows:

```

Class B
    Inherits A

    Overloads Sub MyMethod()
        ArabicConsole.WriteLine("B.MyMethod")
    End Sub

    Shadows Sub MyMethod2()
        ArabicConsole.WriteLine("B.MyMethod2")
    End Sub
End Class

```

عند إنشاء الكائن واستدعاء الطرق، لاحظ ماذا سيجري:

```

Dim obj As New B()

obj.MyMethod() ' B.MyMethod
obj.MyMethod(10) ' A.MyMethod (x)
obj.MyMethod2() ' B.MyMethod2

```

اعلم أنك ستتساءل عن عدم استدعائي للطريقة (x) `obj.MyMethod2` والسبب اني لو حاولت استدعائها ستظهر رسالة خطأ، لأن الكلمة المحجوزة `Shadows` تؤدي إلى تظليل جميع الطرق المعاد تعريفها في الفئة القاعدية باختلاف وسمياتها، وبعبارة أخرى لن تكون هناك إلا طريقة `MyMethod()` واحدة فقط موجودة في الفئة المشتقة.

التصنيف الفرعي Subclassing:

كما رأيت سابقاً، التظليل يمكنك من إعادة قيادة الطريقة في الفئة المشتقة حتى لو لم يسمح مؤلف الفئة القاعدية بذلك، وفي العادة لن تقوم بعمل التظليل إلا عند الحاجة (فمؤلف الفئة القاعدية لن يمنحك من إعادة قيادة الطريقة حتى لا تؤثر على سلوك الفئة القاعدية بشكل سلبي)، يمكنك مثلاً استدعاء الطريقة في الفئة القاعدية دون إجراء أي تعديل عليها في الفئة المشتقة:

```

Class B
    Inherits A

    ...
    ...
    Shadows Sub MyMethod2()
        MyBase.MyMethod2 ()
    End Sub
End Class

```

أما إن قمت بكتابة حرف واحد إضافي من الشيفرة المصدرية في داخل الطريقة السابقة، فاعلم أنك تطبق ما يسمى **التصنيف الفرعي Subclassing** لطريقة الفئة القاعدية:

```
Class B
    Inherits A

    ...

    Shadows Sub MyMethod2()
        If ... Then
            MyBase.MyMethod2 ()
        Else
            ...
        End If
    End Sub
End Class
```

الأعضاء المشتركة Shared Members

كما ذكرت سابقاً، الأعضاء المشتركة لا يمكن إعادة قيادتها، ولا يمكن أيضاً استخدام الكلمات المحجوزة Overrides أو Overridable عليها، ولكنك تستطيع تظليلها باستخدام Shadows:

```
Class BaseClass
    Shared Sub MyMethod()
        ...
    End Sub
End Class

Class DerivedClass
    Inherits BaseClass

    ' يمكنك تظليل الطريقة المشتركة
    Shared Shadows Sub MyMethod()
        ...
    End Sub
End Class
```

نقطة إضافية أخرى، لا يمكنك استخدام MyBase للوصول إلى أعضاء الفئة القاعدية من داخل الأعضاء المشتركة لحظة الاشتقاق الوراثي، حيث سيعترض مترجم اللغة ويظهر لك رسالة خطأ، والحل الوحيد هو استخدام اسم الفئة القاعدية:

```

Class DerivedClass
  Inherits BaseClass

  Shared Shadows Sub MyMethod()

    MyBase.MyMethod ( )      ' رسالة خطأ

    BaseClass.MyMethods ( ) ' استخدم اسم الفئة القاعدية
  End Sub
End Class

```

كلمات محجوزة إضافية

في هذا القسم سأعرض لك ثلاث كلمات محجوزة لم أتطرق إليها سابقاً، اثنتان تتعلق بقابلية الوراثة بين الفئات والثالثة بإعادة القيادة.

الكلمة المحجوزة NotInheritable

لأسبابك الشخصية، قد تمنع أحد المتطفلين من اشتقاق فئاتك التي سهرت الليالي في بنائها، يمكنك بكل سهولة عمل ذلك باستخدام الكلمة المحجوزة NotInheritable عند تعريف الفئة:

```

' لا يمكن اشتقاق هذه الفئة
NotInheritable Class MyClass
  ...
  ...
End Class

```

في اغلب الأحوال، الفئات التي لا تحتوي الا على أعضاء مشتركة يفضل ان تجعلها غير قابلة للوراثة، لديك مثلاً الفئات ArabicConsole و Console لن تتمكن من وراثتها:

```

Class DerivedClass
  Inherits ArabicConsole ' في المضمّن !
  ...
  ...
End Class

```

الكلمة المحجوزة MustInherit

كما هو واضح من اسمها، الكلمة المحجوزة MustInherit تجبر المبرمج على اشتقاقها أولاً ومن ثم إنشاء كائنات من الفئات المشتقة منها:

```
MustInherit Class BaseClass
...
...
End Class
```

لن تستطيع إنشاء نسخة كائن من هذه الفئة مباشرة، بل عليك اشتقاقها أولاً في فئة (كـ DerivedClass مثلاً) ومن ثم إنشاء كائن من تلك الفئة المشتقة:

```
Dim X As New BaseClass() ' رسالة خطأ
Dim X As New DerivedClass() ' ممكن
```

يعرف هذا النوع من الفئات بالفئات المجردة **Abstract Classes**، قد تحتاج إلى الفئات المجردة في حياتك البرمجية إن ردت تكوين فئات لا بد من اشتقاقها ولا يمكن استخدامها مباشرة. مثلاً، في موقعنا شبكة المطورون العرب (dev4arabs.com) عرفنا فئة مجردة تعبر عن وحدة أو سجل:

```
MustInherit Class Item
    Public Title As String
    Public PostedDate As Date
    Public Author As String
    ...
    ...

    Sub Delete()
        ...
    End Sub

    Overridable Sub Update()
        ...
    End Sub

    Overridable Sub AddNew()
        ...
    End Sub
End Class
```

هذه الفئة لا يمكن استخدامها مباشرة حيث أنها بحالتها الراهنة لا ترمز إلى نوع معين من البيانات، فالذي قمنا به هو تعريف فئات أخرى تشتق منها، بعضها يقوم بإعادة قيادة طرقها عند الحاجة:

```

' قسم الشيفرات المصدرية '
Class SourceCode
    Inherits Item
    ...
End Class

' قسم التلميحات '
Class Tip
    Inherits Item
    ...
End Class

' قسم المقالات '
Class Article
    Inherits Item

    Public Introduction As String
    ...
    Public Overrides Sub Update()
    ...
End Sub

    Public Overrides Sub AddNew()
    ...
End Sub
End Class

```

الكلمة المحجوزة MustOverride

بعض الطرق والخصائص في الفئات لا بد من أن يتم إعادة قيادتها حتى تتمكن من محاكاة الفئة المشتقة منها، فمثلاً لو أردنا تعريف طريقة في الفئة Item السابقة لعرض البيانات:

```

MustInherit Class Item
    ...
    Sub ShowDate()
    End Sub
End Class

```

فلا يمكن كتابة شيفرات لعرض البيانات لاختلاف نوع البيانات واختلاف طريقة عرضها في صفحات الموقع، لذلك أضفنا الكلمة المحجوزة MustOverride حتى تجبر مستخدم الفئة من إعادة قيادة الطريقة لحظة اشتقاقها:

```
MustInherit Class Item
...
...
MustOverride Sub ShowDate()
End Class
```

عند استخدامك للكلمة المحجوزة MustOverride فلا تكتب العبارة End أو End Sub أو End Function أسفل الطريقة، ولا التركيب Get ... Set أو End Property عند تعريف خاصية.

ملاحظة

لن تتمكن من استخدام الكلمة المحجوزة MustOverride في الفئة إلا عند استخدامك للكلمة المحجوزة MustInherit لحظة تعريف الفئة، والسبب يبدو منطقيا جدا ولست بحاجة إلى توضيحه.

محددات الوصول

ذكرت في الفصول السابقة ثلاث كلمات محجوزة تستخدم لتحديد قابلية الرؤية Visibility هي: Public، Private، و Friend. بالإضافة إلى هذه الكلمات، توجد كلمتين أخريين هما Protected و Protected Friend إلى هذا الفصل لأنهما تتعلقان بالوراثة. في الفقرات التالية سنرى مدى تأثير هذه الكلمات الجديدة، بالإضافة إلى مراجعة الكلمات السابقة.

قابلية الرؤية للغئات

خمس كلمات محجوزة تستخدم لتحديد قابلية الرؤية للفئة هي:

:Private

الغئات المصروفة بـ Private يمكن الوصول لها من داخل الوعاء الذي عرفت فيه فقط، الوعاء قد يكون -كما ذكرت سابقا- اما وحدة برمجية Module، تركيب من نوع Structure، أو فئة Class، ولن تستطيع الوصول لها في مكان آخر:

```
' غير ممكن
Private Class TestClass
...
End Class
```

```
Module Module1
    ' ممكن
    Private Class TestClass
        ...
    End Class
    ...
End Module
```

بالنسبة للفئات المتداخلة Nested، فلن تتمكن من الوصول لها ان استخدمت Private عند تعريف كائن خارج نطاق وعائها:

```
Class Outer
    Private Class Inner
        ...
    End Class
    ...
End Class

Module Module1
    Sub Main()
        Dim X As New Outer.Inner() ' لا يمكن
        ...
    End Sub
End Module
```

كما ترى، لا يمكن الوصول إلى الفئة Inner إلا من داخل الفئة Outer، أو أي فئة أخرى معرفة داخل Outer:

```
Class Outer
    Private Class Inner
        ...
    End Class

    Class Inner2
        Public X As Inner ' ممكن
        ...
    End Class
    ...
End Class
```

:Friend

استخدامك للكلمة المحجوزة Friend (أو حتى تجاهلها فهي افتراضية) عند تعريف الفئة، يمكنك من استخدام هذه الفئة والوصول لها من مختلف أماكن المشروع الحالي، يمكنك تعريف فئة من النوع Friend من أي مكان ترغبه -حتى وإن كانت داخل وعاء لفئة أخرى:

```

Class Outer ' Friend هنا تعني
    Friend Class Inner
    ...
End Class
End Class

Module Module1
    Sub Main()
        Dim X As New Outer.Inner() ' يمكن جدا
    ...
End Sub
End Module

```

مع ذلك، إن عرفت فئة باستخدام Friend داخل وعاء بمستوى Private فلن تتمكن من الوصول إليها إلا من داخل ذلك الوعاء فقط، فالفئة C التالية:

```

Class A
    Private Class B
        Friend Class C
        ...
    End Class
    ...
End Class

```

لن تتمكن من الوصول لها خارج الفئة B رغم انها على مستوى Friend. المزيد أيضا، ان عرفت فئة داخل فئة أخرى فلن تتمكن من تعريف كائن منها بكتابة اسم الفئة مباشرة حتى لو كانت Friend، اذ يشترط كتابة اسم الفئة الحاضنة لها أولا، فمثلا الفئة B التالية:

```

Class A
    Class B
    ...
    ...
End Class

```

عليك ذكر اسم الفئة الحاضنة لها لحظة تعريف كائن جديد:

```

Dim obj As New A.B() ' هكذا
Dim obj As New B() ' وليس كذا

```

:Public

كل ما ذكرته حول Friend ينطبق على Public فهي مثل Friend تماما، الا ان لها ميزة إضافية حيث يمكنك من الوصول إلى الفئة من خارج المشروع الحالي أيضا. قد تستخدم Public في اغلب الأحوال عند بناء مكتبة فئات Class Library تترجم إلى ملفات DLL مثلا.

:Protected

اما Protected فهي مثل Private بشكل عام (فكل ما ذكرته عن Private سابقا يطبق هنا)، إلا انها تكون قابلة للوصول من الفئات المشتقة فقط، فمثلا الفئة Inner التالية:

```
Class BaseClass
    Protected Class Inner
    ...
End Class
...
End Class
```

لن تستطيع الوصول لها واستخدامها إلا عن طريق فئة مشتقة من BaseClass فقط:

```
Class DerivedClass
    Inherits BaseClass

    Private X As Inner ' ممكن
    ...
End Class

Module Module1
    Sub Main()
        Dim X As Inner ' لا تتعب نفسك
        ...
    End Sub
End Module
```

في المثال السابق، يتوجب علي اشتقاق BaseClass حتى أتمكن من الوصول إلى الفئة Inner، لأن Inner كما أخبرتك Private ولن أستطيع اشتقاقها مباشرة:

```
Class DerivedClass
    Inherits Inner ' بودي ولكن للأسف
    ...
End Class
```

:Protected Friend

يسمح لك محدد الوصول Protected Friend باستخدام الفئة في أي مكان من المشروع الحالي، أي أنه مثل Friend تماما، فكل ما ذكرته عن Friend ينطبق هنا على Protected Friend دون أي مشاكل، والفرق الوحيد بين Friend و Protected Friend هو أن Protected Friend تسمح لك باستخدام الفئة أن تم اشتقاقها من مشروع آخر، فالفئة Inner التالية:

```
Class BaseClass
    Friend Class Inner
    ...
End Class

Protected Friend Class Inner2
    ...
End Class
...
End Class
```

يمكنك استخدامها من أي مكان في المشروع الحالي فقط، اما الفئة Inner2 فتستطيع استخدامها أيضا إن قمت باشتقاقها وراثيا من مشاريع أخرى.

قابلية الرؤية لأعضاء الفئات

تحدثت في الفقرة السابقة عن تأثير الكلمات المحجوزة عند تعريفها مع الفئة، دعنا نلقي الضوء هنا على أعضاء الفئة وبيان مدى تأثيرها.

:Private

كما هو معلوم، الأعضاء المصرحة بـ Private لا يمكن الوصول لها إلا من داخل الفئة فقط، وبالنسبة للحقول Fields فهي ستكون Private بشكل افتراضي:

```
Class TestClass
    Dim X As Integer ' هنا Dim تعني Private
    Private Sub MyMethod()
    ...
End Sub
...
End Class
```

وبالنسبة للأحداث Events على المستوى Private، فلن تتمكن من قنصها الا من داخل الفئة فقط:

```

Class TestClass
    Private Event MyEvent()

    Public Sub MySub()
        AddHandler Me.MyEvent, AddressOf Me.EventHandler
        ...
    End Sub

    Public Sub EventHandler()
        ...
    End Sub
End Class

```

اما الحديث عن الفئات المتداخلة، فيمكنك الوصول إلى أعضاء الفئة الحاضنة من داخل الفئة المحضونة حتى لو كانت على مستوى **:Private**:

```

Class Outer
    Private X As Integer

    Class Inner
        Dim obj As Outer ' عليك إسناد قيمة لهذا المؤشر

        Sub MyMethod()
            ...
            obj.X = 10 ' ممكن
            ...
        End Sub
    End Class
End Class

```

:Friend

مرة أخرى، جميع الأعضاء المعرفة باستخدام **Friend** يمكنك الوصول لها من خارج الفئة (عن طريق مؤشر الكائن الذي ستعرفه)، جميع أعضاء الفئة تكون **Public** وليس **Friend** بشكل افتراضي باستثناء الحقول التي تكون **Private** إن لم تحدد محدد الوصول لها:

```

Class TestClass
    Friend X As Integer

    Friend Sub MyMethod () ' Friend
        ...
    End Sub

    Sub MyMethod2 () ' Public
        ...
    End Sub
    ...

```

```
...
End Class
```

عندما تعرف عضو على مستوى Friend فمن غير المنطقي استخدام فئات أو نوع خاص Private (حيث انه خاص بالوعاء الذي عرف فيه فقط ولن تستطيع ايبصاله للعالم الخارجي):

```
Class Outer
    Friend X As Inner ' ولا في الاحلام

    Private Class Inner
        ...
    End Class
End Class
```

:Public

استخدام Public مع أعضاء الفئات هو مثل استخدام Friend (وجميع مع ذكرته في الفقرة السابقة يطبق أيضا هنا على Public)، ولكن الفرق عند هو قدرة الوصول إلى الأعضاء على مستوى Public من خارج المشروع الحالي، اما Friend فهي محصورة داخل المشروع فقط.

:Protected

أيضا، الأعضاء على مستوى Protected هي أعضاء من النوع Private (وكل ما ذكرته حول Private ينطبق أيضا على Protected أيضا)، ولكنك تستطيع الوصول إليها لحظة اشتقاق الفئة:

```
Class BaseClass
    Protected X As Integer
    ...
End Class
```

يمكنك الوصول إلى الحقل X السابق من أي فئة مشتقة من BaseClass:

```
Class DerivedClass
    Inherits BaseClass

    Sub MyMethod()
        Me.X = 10
    End Sub
End Class
```

قد تعتقد انك تستطيع الوصول إلى المتغير X السابق من خلال مؤشر الكائن الذي تعرفه من الفئة المشتقة، ولكن هذا غير ممكن:

```
Dim obj As New DerivedClass()

obj.x = 10 ' غير ممكن
```

:Protected Friend

الأعضاء على مستوى Protected Friend هي كالأعضاء على مستوى Friend، ولكنها تزيد عنها في إمكانية الوصول لها من فئات مشتقة عرفت خارج المشروع الحالي.

تأثير محددات الوصول على المشيدات

قد يثير اهتمامك معرفة مدى تأثير محددات الوصول السابقة على المشيدات، وان كان لا يثير اهتمامك دعني اريك هذه الفئة:

```
Public Class TestClass
    Private Sub New()
        ...
    End Sub
End Class
```

رغم ان الفئة السابقة Public، الا انك لن تستطيع إنشاء نسخة منها، فالسطر التالي سيظهر رسالة خطأ:

```
Dim obj As New TestClass()
```

علي توضيح فرق هام هنا، وهو ان الفئة TestClass يمكن استخدامها في البرنامج دون مشاكل (كأن تجعلها وسيطة لاحد الاجراءات)، ولكنك لن تستطيع إنشاء نسخة كائن منها بنفسك (باستخدام New مثلا). يعرف هذا النوع من الفئات بالفئات غير قابلة للإشياء **Not Creatable Class**.

المشيدات من النوع :Private

كما ذكرت في الصفحة السابقة، ان كان مشيد الفئة من النوع Private فلن تتمكن من انشاء نسخة كائن جديدة من الفئة مهما كان نوعها.

المشيدات من النوع Friend:

اما ان كان المشيد من النوع Friend، فيمكن انشاء نسخة كائن جديدة من الفئة في المشروع الحالي فقط.

المشيدات من النوع Public:

وهو النوع الافتراضي للمشيدات، بحيث يمكنك انشاء نسخة كائن جديدة من الفئة حتى في المشاريع الأخرى.

المشيدات من النوع Protected:

ان كان المشيد من النوع Protected فيمكنك انشاء نسخة كائن من الفئة من داخل الفئة المشتقة فقط، فالفئة الحالية:

```
Public Class BaseClass
    Protected Sub New()
        ...
    End Sub
End Class
```

لن تستطيع انشاء نسخة كائن جديدة منها الا من داخل فئة مشتقة فقط:

```
Class DerivedClass
    Inherits BaseClass

    Public x As New BaseClass()
    ...
End Class
```

المشيدات من النوع Protected Friend:

مثل تأثير المشيدات من النوع Friend، ولكن ستكون عملية انشاء نسخة كائن جديدة ممكنة أيضا للفئات المشتقة في مشاريع أخرى.

ملاحظة

في جميع الفقرات السابقة التابعة لهذا القسم من الفصل، ذكرت ان الفرق بين Public و Friend يظهر عند الخروج عن إطار المشروع الحالي، رغم ان العبارة صحيحة تقريبا إلا أنها غير دقيقة، فالفرق يظهر عند الخروج عن إطار المجمع Assembly الحالي، حيث أن المجمع قد يحتوي على أكثر من مشروع كما ستري لاحقا في الفصل الحادي عشر **المجمعات Assemblies**.

قدمت في هذا الفصل كل ما أود ذكره حول الوراثة والاشتقاق الوراثي بين الفئات، لتكون مستوعباً تماماً وجاهزاً لاستخدام مكتبة فئات إطار عمل .NET Framework. وعليك معرفة ان مبدأ الوراثة في لغات OOP -بصفة عامة- من اعقد المبادئ التي تواجه المبرمجين، وهذا هو كل ما استطعت فعله لتوضيح الوراثة لك. الفصل التالي **الواجهات، التفويض، والمواصفات** سيكون آخر فصل ننهي الجزء الأول **الأساسيات** من هذا الكتاب.

الواجهات، التفويض، والمواصفات

ونحن على مشارف الانتهاء من تعلم أساسيات لغة البرمجة Visual Basic .NET، تبقى لدينا مجموعة من المواضيع المتفرقة والتي يتحتم على ذكرها من منطلق الشمولية للإلمام بأساسيات لغة البرمجة Visual Basic .NET.

سأختم معك الجزء الأول من هذا الكتاب بالحديث عن مواضيع متعددة كالواجهات Interfaces وطريقة تطبيق مبدأ تعدد الواجهات Polymorphism، كما سأطرق أيضا إلى الإجراءات المفوضة Delegates والتي تعطيك مرونة كبيرة في استدعاء الإجراءات، واختتم الفصل بالتلميح إلى موضوع المواصفات Attributes لننتهي بذلك مرحلة تعلم أساسيات لغة البرمجة Visual Basic .NET.

الواجهات

في البداية دعني أوضح لك ما المقصود بكلمة واجهة في لغات OOP. الواجهة Interface هي مجموعة من الطرق والخصائص والأحداث التي تصف فئة معينة، فالفئة التالية:

```
Class Person
    Event Die()

    Public Property Name() As String
    ...
    ...
End Property

Sub Move()
    ...
    ...
End Sub
End Class
```

تحتوي على واجهة متمثلة في حدث باسم Die، خاصية باسم Name، وطريقة باسم Move(). لاحظ ان كلمة الواجهة تطلق على أسماء الأعضاء فقط وليس وظائفها (أي ليس سلوكها أو الشيفرات التي تحتويها).

من المبادئ الأساسية في لغات OOP هو مبدأ تعدد الواجهات Polymorphism، وهو احتواء الفئة الواحدة على أكثر من واجهة تصفها، ليصل الأمر إلى ان نتشارك مجموعة فئات في واجهات موحدة فنقول: أسماء متشابهة لكن إنجازات مختلفة Same names but different implementation.

العديد من الفوائد التي تجنيها من تطبيق واستخدام مبدأ تعدد الواجهات Polymorphism، لعل أبرزها اختصار جمل الشرط في برنامجك، فتخيل ان لدينا فئة تمثل مستند نصي TextFile، وفئة أخرى تمثل مستند منسق RTFFile، وفئة أخرى تمثل مستند صفحة ويب HTMLFile، وفئة رابعة تمثل مستند تنسيق البيانات XMLFile، وفئة أخيرة تمثل صورة PictureFile، ستجد ان برمجة وإنجاز كل فئة من هذه الفئات سيختلف تماماً عن الفئة الأخرى، ولكن الواجهات التي تحتويها متشابهة ومشاركة، فيمكن تعريف خاصية تمثل اسم الملف FileName أو حجم الملف FileSize، وطريقة تؤدي إلى عملية الحفظ Save() أو الحذف Delete()، ويمكن تطبيق جميع هذه الواجهات على الفئات المختلفة.

فمثلاً لو قمنا بتطوير إجراء يقوم بحفظ الملف النصي فستجعل الوسيطة تستقبل كائن من النوع TextFile:

```
Sub SaveTextFile(ByVal FileObject As TextFile)
    ...
    FileObject.Save
    ...
End Sub
```

ونفس الشيء سنتفعله مع الأنواع الأخرى من الملفات:

```
Sub SaveRTFFile(ByVal FileObject As RTFFile)
    ...
    FileObject.Save
    ...
End Sub

Sub SaveHTMLFile(ByVal FileObject As HTMLFile)
    ...
    FileObject.Save
    ...
End Sub
```

ليس هذا فقط، بل أنه عند استدعاء إجراء الحفظ ستجد أنه يتوجب عليك كتابة الجمل الشرطية في كل مرة نود فيها فعل ذلك للتحقق من نوع الملف، وعلى ضوء المقارنة تقوم باستدعاء فئة الملف المعنية لتقوم بالحفظ، كما أن عليك أيضاً الإعلان عن جميع الكائنات التي تمثل الفئات المختلفة لإرسالها إلى الإجراء المختص:

```
Dim TextFileObject As New TextFile
Dim RTFFileObject As New RTFFile
Dim HTMLFileObject As New HTMLFile

...

Select Case FileType
    Case 1
        TextFileObject.FileName = "xxxx"
        SaveTextFile ( TextFileObject )
        ...
    Case 2
        RTFFileObject.FileName = "xxxx"
        RTFRTFFile ( RTFFileObject )
        ...
    Case 3
        HTMLFileObject.FileName = "xxxx"
        HTMLTextFile ( HTMLFileObject )
        ...
    ...
End Sub
```

والآن لنفترض انه ظهرت الحاجة لإضافة فئة جديدة لنوع معين من التسيقات BinaryFile مثلاً، فإن ذلك يفرض عليك ان تضيف جميع الاستدعاءات والإجراءات والإعلانات والشروط التي ستتعامل مع هذه الفئة في كل أنحاء البرنامج، وتقوم بتكرار كل ما فعلته في الفئات السابقة. ولكن مع تعدد الواجهات فإن الأمر مختلف تماماً، فكل ما سنفعله هنا هو القيام بتعريف واجهة نسميها مثلاً IFile ونعرف إجراء يستقبل كل أنواع الكائنات شريطة احتوائه على الواجهة IFile:

```
Sub SaveFile(ByVal FileObject As IFile)
    ...
    FileObject.Save ()
    ...
End Sub
```

لا يقتصر الإجراء السابق على استقبال الأنواع التي ذكرناها سابقا (TextFile، RTFFile، HTMLFile، ... الخ) بل يمكنه ان يستقبل كل الأنواع الأخرى والتي قد تحتوي يوما من الأيام على الواجهة IFile.

بناء واجهة

بعد ان أوضحت لك الفكرة من الواجهات، لنبدأ ببناء واجهة تحمل الاسم IFile، تحتوي على جميع الأعضاء التي نود من الفئات الأخرى استخدامها. لتعريف واجهة في Visual Basic .NET، استخدم التركيب Interface ... End Interface:

```
Interface IFile
    ' خصائص
    Property FileName() As String
    Property FileSize() As Integer
    ...

    ' طرق
    Sub Save()
    Sub Save(ByVal TargetFile As String)
    Sub Delete()
    ...
End Interface
```

كما ترى في التركيب السابق، الواجهة لا تحتوي على أية شيفرات برمجية فهي مجرد واصفة لأسماء ووسيطات الأعضاء فقط، فلا تفكر في وضع عبارات كـ End Function، End Sub، أو End Property.

ملاحظة

جرى العرف سابقا عند مبرمجي OOP باستخدام الحرف I قبل اسم الواجهة، وهو نفس الأسلوب الذي تقترحه مستندات .NET. مع استخدام PascalCase لكتابة اسم الواجهة.

بالنسبة لقابلية الرؤية Visibility للواجهات، فهي افتراضيا Friend ويمكنك استخدام محددات الوصول الأخرى كما تفعل مع الفئات تماما:

```
' Friend
Interface IFile
...
End Interface

' Public
Public Interface IView
...
End Interface

Module Module1
    ' Private
    Private Interface ITool
        ...
    End Interface

    Sub Main()
        ...
    End Sub
End Module
```

مع ذلك، لا يمكنك استخدام محددات الوصول مع أعضاء الواجهة، فجميع أعضاء الواجهات قابلة رويتها Public دائما وأبدا:

```
Interface ITest
    ' لا يمكن استخدام محددات الوصول
    ' مع أعضاء الواجهة
    Public Sub MyMethod ()
    Private Sub YourMethod ()
    ...
    ...
End Interface
```

المزيد أيضا، يمكن للواجهات ان تكون متداخلة Nested:

```
Interface IView
    Interface IWindow
        ...
    End Interface

    Interface IWEB
        ...
    End Interface
    ...
End Interface
```

اخيراً، الطرق، الخصائص، والأحداث هي التي يمكن كتابتها في الواجهات، أما الحقول فغير ممكنة:

```
Interface IMyInterface
    X As Integer ' بوندنا ولكن لاسف
...
End Interface
```

تضمين الواجهة

بعد تعريفك للواجهة، يمكنك تضمينها في أي فئة باستخدام الكلمة المحجوزة **Implement**:

```
Class TextFile
    Implements IFile ' تضمين الواجهة
...
...
End Class
```

الشرط الأساسي لتضمين الواجهة في الفئة الحالية هو تعريف جميع أعضاء الواجهة وإلا ستظهر لك رسالة خطأ:

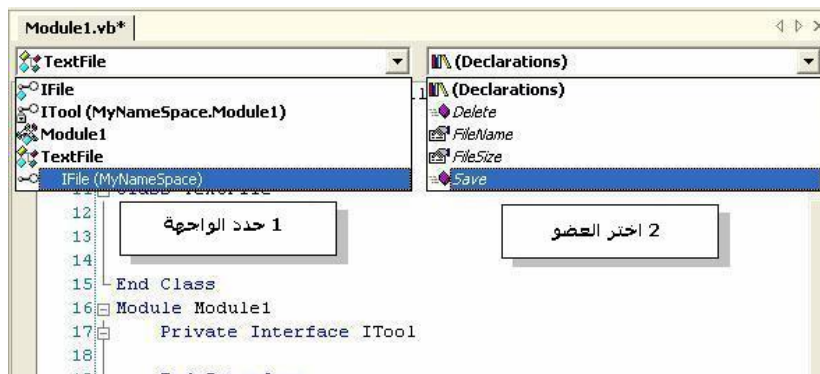
```
Class TextFile
    Implements IFile

    ' FileName الخاصية
    Property FileName() As String Implements IFile.FileName
        Get
            ...
        End Get
        Set(ByVal Value As String)
            ...
        End Set
    End Property

    ' Delete() الطريقة
    Sub Delete() Implements IFile.Delete
        ...
    End Sub

    ' عليك اضافة جميع الاعضاء الاخرى
    ...
    ...
End Class
```

وبدلاً من تعريف أعضاء الواجهات يدوياً، يمكنك الاستعانة بمحرر الشيفرات لعمل ذلك نيابة عنك، حدد الواجهة من القائمة العلوية اليسرى ومن ثم اختر العضو المراد تعريفه من القائمة العلوية اليمنى (شكل 5-1).



شكل 5-1: الاستعانة بمحرر الشيفرة لكتابة أعضاء الواجهة.

عند تعريف أعضاء الواجهة، فلا بد من توافق وسيطات الطرق، الأحداث، والخصائص مع نفس الوسيطات الموجودة في الواجهة، ولابد من توافق نوع القيم التي تعود بها الدوال Functions. المزيد أيضاً، قابلية القراءة والكتابة (اقصد استخدام الكلمات ReadOnly و WriteOnly) للخصائص، لابد وان تكون مطابقة لتلك الموجودة في الواجهة. فالتضمينات الموجودة في الفئة TestClass بالصفحة التالية خاطئة 100%:

```
Interface IMyInterface
    ' خاصية للقراءة فقط
    ReadOnly Property BirthDate() As Date

    ' دالة تعود بقيمة حرفية
    Function MyMethod() As String
End Interface
```

```

Class TestClass
    Implements IMyInterface

    ' قابلية القراءة والكتابة غير متوافقة
    Public WriteOnly Property BirthDate() As Date Implements _
        IMyInterface.BirthDate

        Set(ByVal Value As Date)
        ...
    End Set
End Property

    ' القيمة التي تعود بها الدالة غير متوافقة
    Public Function MyMethod() As Integer Implements _
        IMyInterface.MyMethod

        ...
    End Function
End Class

```

بالنسبة للواجهات المتداخلة، فيمكنك تضمينها كما تفعل مع الواجهات العادية، ولكن لا تنسى كتابة اسم الواجهة الحاضنة لها أولاً:

```

Interface IView
    Interface IWindow
        ...
    End Interface

    Interface IWEB
        ...
    End Interface
    ...
End Interface

Class TestClass
    Implements IView.IWindow ' لا تنسى كتابة اسم الواجهة الحاضنة أولاً
    ...
End Class

```

أما إن استخدمت الواجهة الحاضنة:

```

Class TestClass2
    Implements IView
    ...
End Class

```

فالمؤكد ان الفئة السابقة لن تحتوي إلا على الواجهة IView فقط، ولا تعتقد للحظة ان الواجهات المحصورة بها (IWeb و IWindow) قد تم تضمينها هي الأخرى. أخيراً، يمكن للفئة الواحدة ان تحتوي على أكثر من واجهة:

```
Class ManyInterfaces
    Implements IFile
    Implements IView
    Implements IView.IWindow
    Implements IView.IWEB
...
...
End Class
```

عند تعارض اسماء الأعضاء في أكثر من واجهتين، فلا بد من تغيير اسمها لحظة كتابتها في الفئة، فمثلاً لو اكتشفت أن الطريقة MyMethod() موجودة في الواجهتين السابقتين IView و IFile، فيمكنك تمييزهما بوضع رقم:

```
Class ManyInterfaces
    Implements IFile
    Implements IView
    ...
    ...

    Sub MyMethod() Implements IFile.MyMethod
        ...
    End Sub

    ' لابد من تغيير اسم الطريقة '
    Sub MyMethod2() Implements IView.MyMethod
        ...
    End Sub
    ...
    ...
End Class
```

من المثال السابق، يتضح لنا ان مترجم .NET Visual Basic لا يهتم باسم العضو بقدر ما يهتم بالاسم الذي يلي الكلمة المحجوزة Implements لحظة تعريفه، لنستنتج من ذلك انه يمكننا تغيير اسم الطرق السابقة:

```

Class ManyInterfaces
    Implements IFile
    Implements IView
    ...
    ...

    Sub MethodFromFile () Implements IFile.MyMethod
        ...
    End Sub

    Sub MethodFromView () Implements IView.MyMethod
        ...
    End Sub
    ...
    ...
End Class

```

الوصول إلى الواجهة

بافتراض ان فئات الملفات السابقة (HTMLFile، RTFFFile، TextFile) قد تم تضمين الواجهة IFile بها، فان باستطاعتنا تعريف متغير من النوع IFile وإسناد اي قيمة له من النوع HTMLFile، RTFFFile، TextFile... الخ:

```

Dim textFileObject As New TextFile()
Dim file As IFile

' يمكن عمل ذلك
file = textFileObject
file.FileName = "MyFile.TXT"
...
...

```

كما يمكن تعريف إجراء يستقبل وسيطة من النوع IFile:

```

Sub DoSave(ByVal fileObject As IFile)
    ...
    fileObject.Save ()
    ...
End Sub

```

لنتمكن من إرسال اي قيمة له من النوع HTMLFile، RTFFFile، TextFile.... الخ:

```
Dim htmlFileObject As New HTMLFile()
...
...
DoSave (htmlFileObject)
...
...
```

وراثة الواجهات

عند الحديث عن وراثه الواجهات، يمكننا ان ننظر إليها من منظورين مختلفين، المنظور الأول عند اشتقاق فئة قاعدية تحتوي على واجهة:

```
Interface IMyInterface
    Sub MyMethod()
End Interface

Class BaseClass
    Implements IMyInterface

    Sub MyMethod() Implements IMyInterface.MyMethod
        ...
    End Sub
End Class
```

عندما تشتق الفئة BaseClass السابقة، عليك معرفة ان الواجهة IMyInterface المضمنة بها سيتم اشتقاقها أيضا:

```
Class DerivedClass
    Inherits BaseClass

    ...
    ...
End Class
```

لنتعامل مع الكائن من الفئة DerivedClass كما نتعامل مع الكائن BaseClass:

```
Dim DerivedObject As New DerivedClass()
Dim X As IMyInterface

X = DerivedObject

' واجهة من الفئة القاعدية
X.MyMethod()
```

اما المنظور الثاني فهو يتعلق بوراثنة الواجهات، وهو توارث الواجهات فيما بينها باستخدام الكلمة المحجوزة Inherits أيضا:

```
Interface IBaseInterface
    Sub MyMethodInBase()
End Interface

Interface IDerivedInterface
    Inherits IBaseInterface

    Sub MyMethodInDerived()
End Interface
```

عند تضمين الواجهة IDerivedInterface في اي فئة، فيتوجب عليك تعريف جميع الأعضاء التابعة للواجهة القاعدية أيضا:

```
Class TestClass
    Implements IDerivedInterface

    ' الواجهة القاعدية '
    Sub MyMethodInBase() Implements IDerivedInterface.MyMethodInBase
        ...
    End Sub

    ' الواجهة المشتقة '
    Sub MyMethodInDerived() Implements
        IDerivedInterface.MyMethodInDerived
        ...
    End Sub
End Class
```

واجهات من إطار عمل .NET Framework

في الفصل الثالث الفئات والكائنات تعرفنا على الواجهة IDisposable، وذكرت حينها انك تضمناها في الفئة لتعرف الإجراء Dispose() لمحاكاة المهذمات Destructors في الفئة:

```
Class TestClass
    Implements IDisposable

    Public Sub Dispose() Implements System.IDisposable.Dispose
        ...
    End Sub
    ...
End Class
```

توجد مئات الواجهات الأخرى والمعرفة في مكتبة فئات إطار عمل .NET Framework. إلا أنني سأحاول في هذا القسم عرض بضعة واجهات هامة تستخدمها في معظم برامجك ومشاريعك بـ Visual Basic .NET.

الواجهة IComparable

عملية إجراء المقارنة بين عددين تعتمد على قيمة العدد، ويمكنك استخدام معاملات المقارنة (>، <، =>... الخ)، ويمكن أيضا استخدام علامات المقارنة بين القيم الحرفية Strings، حيث سيعتمد المترجم في هذه الحالة على المقابل الرقمي للحرف في جداول المحارف ASCII أو UNICODE حسب الحالة:

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

If X > 5 Then ...
If Y < "أحمد" Then ...
```

ولكن عند الحديث عن أنواع البيانات الأخرى والتي تصممها بفئاتك الخاصة، فلا يمكن تحديد أيهما أكبر وإيهما أصغر، فلو عرفت كائنين من فئة Person فأنت تحدد بنفسك ما هي القيم التي تريد الاعتماد عليها لحظة المقارنة، لذلك سنقوم بتضمين الواجهة IComparable في الفئة لتعرف طريقتها الوحيدة () CompareTo والتي تعود بالقيمة -1 إذا كان الكائن الحالي أصغر من الكائن المرسل، 0 في حالة تساوي الكائنين، أو 1 ان كان الكائن الحالي أكبر من الكائن المرسل:

```
Class Person
    Implements IComparable

    Public Name As String

    Function CompareTo(ByVal obj As Object) As Integer Implements _
        System.IComparable.CompareTo

        Dim tempObj As Person = CType(obj, Person)

        ' الكائن الحالي اصغر من الكائن المرسل '
        If Me.Name < tempObj.Name Then
            Return -1
        ' الكائن الحالي اكبر من الكائن المرسل '
        ElseIf Me.Name > tempObj.Name Then
            Return 1
        End If
    End Function
End Class
```

```

        الكائنات متساويان
    Else
        Return 0
    End If
End Function
End Class

```

في المثال السابق، اعتمدت على خاصية الاسم Name لتحديد ناتج المقارنة بين الكائنين، قد تغير أنت رأيك وتعتمد على خاصية العمر Age مثلا في فئاتك الخاصة:



```

Class Person
    Implements IComparable
    Public Age As Integer

    Function CompareTo(ByVal obj As Object) As Integer Implements _
        System.IComparable.CompareTo

        Dim tempObj As Person = CType(obj, Person)

        ' الاعتماد على خاصية العمر Age
        If Me.Age < tempObj.Age Then
            Return -1
        ElseIf Me.Age > tempObj.Age Then
            Return 1
        Else
            Return 0
        End If
    End Function
End Class

```

والآن فنتنا Person تحتوي على الواجهة IComparable لذلك نستطيع إرسال الكائنات المنشئة من هذه الفئة إلى جميع إجراءات فئات .NET Framework الأخرى (والتي تتطلب الواجهة IComparable طبعا). فمثلا الطريقة Sort() (والتابعة للفئة Array) يمكن ان تستقبل هذا النوع:



```

Dim PersonObject(10) As Person
Dim Counter As Integer

For Counter = 0 To UBound(PersonObject)
    PersonObject(Counter) = New Person()
Next

PersonObject(0).Age = 33
PersonObject(1).Age = 50
PersonObject(2).Age = 14

```

```
...
...
' الطريقة Array.Sort تقبل اي وسيطة تحتوي
' على الواجهة IComparable
Array.Sort(PersonObject)

For Counter = 0 To UBound(PersonObject)
    ArabicConsole.WriteLine(PersonObject(Counter).Age)
Next
```

الواجهة ICloneable

عند التعامل مع المتغيرات المرجعية Reference Type Variables، فقد أخبرتك أكثر من مرة ان عملية إسناد القيم بين متغيرين لا تقوم بإنشاء نسخة من المتغير وإسنادها إلى الآخر، وإنما تقوم بجعل كلا المتغيرين يشيران إلى نفس الكائن:

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

...
...

' لا تقوم بإنشاء نسخة جديدة وإنما
' جعل كلا المتغيرين يشيران إلى نفس الكائن
Ali = Turki
```

معظم الفئات في عالم NET. تقوم بتضمين الواجهة ICloneable لتعريف طريقتها الوحيدة Clone() والتي تقوم بنسخ الكائن وإنشاء نسخة جديدة منه وإسنادها إلى متغير آخر، هذا مثال لتعريف الطريقة في فئة:

```
Class Person
    Implements ICloneable

    Public Name As String
    Public Age As Integer

    Function Clone() As Object Implements ICloneable.Clone
        ' إنشاء كائن مؤقت
        Dim tmpObject As New Person()

        ' نسخ قيم الكائن الحالي إلى الكائن
        ' المؤقت يدويا
        tmpObject.Name = Me.Name
        tmpObject.Age = Me.Age
```

```

        العودة بالكائن المنشئ للتو '
        Return tmpObject
    End Function
End Class

```

وبدلاً من قيامك بإنشاء الكائن ونسخ خصائصه يدوياً (حيث أنه يمكن أن توجد متغيرات سكونية Static داخل الطرق والتي لن تتمكن من نسخها بنفسك)، فيمكنك استخدام الطريقة MemberwiseClone() وهي مشتقة من الفئة System.Object (أعود للذكر بأن جميع البيانات والأنواع وكل شيء في عالم NET. مشتقة وراثياً من الفئة System.Object):

```

Class Person
    Implements ICloneable

    Public Name As String
    Public Age As Integer

    Function Clone() As Object Implements ICloneable.Clone
        العودة بنسخة جديدة من الكائن الحالي '
        Return Me.MemberwiseClone()
    End Function
End Class

```

والآن يمكنك استخدامها بهذا الشكل:

```

Dim Turki As New Person()
Dim Ali As Person

Turki.Name = "تركي"

' نسخ الكائن
Ali = CType(Turki.Clone, Person)

Ali.Name = "علي"

' تحقق ان الكائن الاول لم يتم تغيير قيمته '
ArabicConsole.WriteLine(Turki.Name)

```

كما ترى في الشيفرة السابقة، اضطررت إلى استخدام المعامل CType بافتراض ان العبارة Option Strict On مفعلة في الملف الحالي أو على مستوى المشروع، والسبب ان الطريقة Clone() تعود بقيمة من النوع Object. قد يكون استخدام المعامل CType غير محبذ

لمستخدمي فئاتك، لذلك اقترح عليك تغيير اسم الطريقة Clone() في الفئة واجعلها مخفية، ومن ثم عرف طريقة أخرى لتحل محلها:

```
Class Person
    Implements ICloneable
    ...
    ...

    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
```

هنا سيتمكن مستخدم الفئة من استدعاء الطريقة Clone() دون الحاجة لاستخدام المعامل CType:

```
Ali = Turki.Clone
```

وقبل ان اختتم فقرة الواجهة **ICloneable**، دعني أذكرك وأنبهك بان عمليات النسخ التي قمنا بها سابقاً (باستخدام الطريقة MemberwiseClone()) توجد بها مشكلة خطيرة جداً، ولم اعرضها هنا لان حلها يتم عن طريق ما يعرف بالتسلسل. لذلك، لا تتبع الأسلوب السابق في نسخ الكائنات بمشاريعك الحالية حتى تصل إلى الفصل التاسع تسلسل الكائنات **Object Serialization**. اللهم إني قد بلغت، اللهم فاشهد.

الواجهتان IEnumerable و IEnumerator

ذكرت في الفصل الثاني لغة البرمجة أنك تستطيع تطبيق الحلقة Next ... For Each على المصفوفات Arrays أو المجموعات Collections، وبما أننا نتحدث الآن بلغة الواجهات Interfaces فدعني أكون أكثر دقة معك، وأخبرك ان الحلقة For Each يمكن تطبيقها أيضاً على جميع أنواع البيانات الأخرى التي تدعم الواجهتين IEnumerable و IEnumerator. تعمل الواجهتان IEnumerable و IEnumerator جنباً إلى جنب لتمكين المبرمجين من استخدام الحلقة For Each مع فئاتك، وهذا مثال عملي يشرح كيفية استخدام هاتين الواجهتين، وسنبدأ أولاً بتصميم الفئة SplitString والتي تقوم بتقسيم الجملة إلى كلمات، تحتوي هذه الفئة على الخاصية Sentence والتي تمثل الجملة المراد فصل كلماتها:



```

Class SplitString
    Private currentPosition As Integer = 0

    ' خاصية تمثل الجملة المراد فصل كلماتها
    Private m_Sentence As String

    Property Sentence() As String
        Get
            Return m_Sentence
        End Get
        Set(ByVal Value As String)
            m_Sentence = Value
            Me.Reset()
        End Set
    End Property
End Class

```

الخطوة التالية هي تضمين الواجهتين IEnumerable و IEnumerator في الفئة السابقة:



```

Class SplitString
    Implements IEnumerable
    Implements IEnumerator
    ...
    ...
End Class

```

بالنسبة للواجهة IEnumerable، فهي لا تحتوي إلا على طريقة واحدة GetEnumerator() سيتم استدعاؤها بمجرد تنفيذ حلقة For Each، وهي تعود بقيمة لأي كائن يحتوي على الواجهة الأخرى IEnumerator. يكفي كتابة الأمر Return Me لتنفيذها:



```

Class SplitString
    ...
    ...
    Private Function GetEnumerator() As IEnumerator Implements _
        IEnumerable.GetEnumerator
        Return Me
    End Function
End Class

```

اما الواجهة IEnumerator فهي تحتوي على طريقتين وخاصية واحدة، الطريقة الأولى تسمى MoveNext() يتم استدعائها في كل دورة من دورات حلقة التكرار For Each، ويفترض ان تعود بالقيمة True إن بقي عنصر تالي و False إن لم يتبقى اي شيء:



```
Class SplitString
...
...
Private Function MoveNext() As Boolean Implements
IEnumerator.MoveNext
    If Me.currentPosition > Me.Sentence.Length - 1 Then
        Me.Reset()
        Return False
    Else
        Return True
    End If
End Function
End Class
```

والطريقة الثانية هي Reset() والغرض منها إعادة المؤشر الداخلي في الفئة إلى أول عنصر فيها حتى تتمكن من البدء باستخدام الحلقة For Each مرة أخرى:



```
Class SplitString
...
...
Private Sub Reset() Implements IEnumerator.Reset
    Me.currentPosition = 0
End Sub
End Class
```

اما الخاصية فهي تحمل الاسم Current والغرض منها العودة بالقيمة الحالية في الحلقة، لا تنسى ان هذه الخاصية للقراءة فقط ReadOnly:



```
Class SplitString
...
...
Private ReadOnly Property Current() As Object Implements _
IEnumerator.Current

    Get
        Dim counter As Integer
        Dim tmpLength As Integer = 0
```

```

        For counter = Me.currentPosition To Me.Sentence.Length - 1
            If Me.Sentence.Chars(counter) = " " Then
                Exit For
            Else
                tmpLength += 1
            End If
        Next
        Current = Me.Sentence.Substring(Me.currentPosition,
tmpLength)
        Me.currentPosition += tmpLength + 1
    End Get
End Property
End Class

```

والآن يمكنك تطبيق الحلقة For Each على الفئة السابقة SplitString بمرونة كبيرة، لتتمكن من كتابة شيئاً مثل:



```

Dim testObject As New SplitString()
Dim x As String

testObject.Sentence = "سيتم فصل كلمات هذه الجملة في سطور مستقلة"

For Each x In testObject
    ArabicConsole.WriteLine(x)
Next

```

مخرجات الشيفرة السابقة ستكون كالتالي:

```

سيتم
فصل
كلمات
هذه
الجملة
في
سطور
مستقلة

```

ملاحظة

الهدف من إنشاء الفئة SplitString هو تعليمي بحث، أردت أن أوضح من خلاله كيفية تطبيق الواجهتين IEnumerable و IEnumerator، فلا تحاول استخدامها في برامجك الجديدة، حيث يوفر لك إطار عمل .NET Framework فئات أفضل وأسرع بكثير لتقسيم النصوص. نقطة أخرى حول تضمين الواجهتين IEnumerable و IEnumerator، من المفضل ألا تقوم بتضمين هاتين الواجهتين في فئة واحدة، فيفضل تضمين الواجهة IEnumerable في فئة حاضنة، والواجهة IEnumerator في فئة محضونة بها، وذلك لتصميم كائني توجه OOP أفضل.

التفويض

يقصد بكلمة **التفويض Delegates**: بتفويض عملية استدعاء الإجراء إلى متغير يشير إلى ذلك الإجراء، مما يعطيك مرونة كبيرة في اختيار أوقات استدعاء الإجراءات، كما يختصر عليك الكثير من الشيفرات المصدرية والتي تستدعي هذه الإجراءات، فلو كان لدينا الإجراءات التالية:

```
Sub MySub (ByVal X As Integer)
    ...
End Sub

Sub YourSub (ByVal X As Integer)
    ...
End Sub
```

فإنه بإمكاننا استدعائها عن طريق متغير يمثل مؤشر إليها:

```
' سيتم استدعاء الإجراء الذي يشير له المتغير DlgT
' كما ستُرسل إليه قيم الوسيطات
DlgT.Invoke (100)
```

المزيد أيضاً، تستطيع إرسال هذه المؤشرات كوسيلة إلى إجراء آخر وتمكن الإجراء من استدعائها:

```
Sub ShowError(ByVal delegatePointer As OneParameter)
    delegatePointer.Invoke("حدث خطأ في العملية")
End Sub
```

تفيدك الطريقة السابقة كثيراً في اختصار تكرار الشيفرات المتشابهة، حيث ان الإجراء السابق قد يظهر رسالة الخطأ اما على شكل نافذة Message Box، في شاشة Console، عرضها على متصفح Browser (ان كنت تصمم موقعاً)، أو الكتابة في ملف... الخ، دون الحاجة لاعادة استدعاء الإجراء السابق من جديد. ميزة أخرى أكثر إثارة-سترى تطبيقاً عملياً عليها قريباً- هي إمكانية استدعاء إجراءات الفئات (أعضاء الفئات) من داخل الإجراءات المشتركة Shared دون مشاكل.

ان كنت من مبرمجي لغة C\C++ فقد يخطر ببالك مؤشرات الدوال **Function Pointers** والتي تمكنك من استدعاء الدوال عن طريق مؤشراتنا. بشكل مبسّط، مؤشرات الدوال في لغة C\C++ شبيهة بالإجراءات المفوضة في لغة .NET. Visual Basic ولكنها تختلف في ميكانيكية عملها وصيغ كتابتها، حيث ان الإجراءات المفوضة في .NET. Visual Basic أكثر أماناً من مؤشرات الدوال في لغة C\C++ وذلك لانها تحصر نوعية الإجراءات بحيث تتوافق مع وظيفتها. نقطة أخرى، الإجراءات المفوضة في .NET. Visual Basic يمكن ان تتبع لإجراءات ستاتيكية Static Procedures أو تابعة لفئات Instance Procedures، بينما مؤشرات الإجراءات في لغة C\C++ لا يمكن تطبيقها الا مع الإجراءات الستاتيكية فقط.

الإجراءات الستاتيكية

اقصد بعبارة الإجراءات الستاتيكية Static Procedures الإجراءات (سواء كانت Subs أو Functions) التي تعرفها في الفئات Classes بحيث تكون مشتركة (باستخدام الكلمة المحجوزة Shared)، أو التي تعرفها في الوحدات البرمجية Modules. عند تفويض الإجراء إلى مؤشر، عليك اولاً القيام بتعريف نوع مؤشر الإجراء باستخدام الكلمة المحجوزة Delegates:



```
Module Module1
    ' تعريف نوع مؤشر الإجراء لابد ان يكون
    ' على مستوى الوحدة البرمجية
    Delegate Function OneParameter(ByVal X As Integer) As Integer

    Sub Main()
        ...
        ...
    End Sub
    ...
    ...
End Module
```

ولنفترض الآن أن لدينا الإجرائين التاليين:



```
Function Abs(ByVal x As Integer) As Integer
    If x < 0 Then
        Return -x
    Else
        Return x
    End If
End Function

Function Square(ByVal x As Integer) As Integer
    Return x * x
End Function
```

ونريد تفويضها إلى أي مؤشر من النوع OneParameter (الذي عرفناه سابقاً باستخدام الكلمة المحجوزة Delegate) في أي وقت بالطريقة التالية:



```
Sub Main ()
    ' تفويض الإجراء Abs ( )
    Dim Dtgt As New OneParameter(AddressOf Abs)
    ...
    ...
End Sub
```

الآن يمكنك استدعاء الإجراء Abs() عن طريق هذا المؤشر باستدعاء الطريقة Invoke() وإرسال الوسائط المطلوبة:



```
Sub Main ()
    Dim Dtgt As New OneParameter(AddressOf Abs)

    ' استدعاء الإجراء
    ArabicConsole.WriteLine( Dtgt.Invoke(-5) ) ' 5
End Sub
```


تستطيع الآن تفويض الإجراء الآخر Square() في أي وقت بإسناد عنوانه (باستخدام الكلمة المحجوزة AddressOf) إلى المؤشر Dtgt السابق، وعملية الاستدعاء ستتم بنفس الطريقة :Invoke()



```
Sub Main ()
    ...
    ...
    ' تفويض الإجراء Square ( )
    Dtgt = AddressOf Square
    ArabicConsole.WriteLine( Dtgt.Invoke(-5) ) ' 25
End Sub
```

المزيد أيضا، تستطيع تعريف إجراء يستقبل وسيطة من النوع OneParameter لتستدعي الإجراءات التي يشير إليها الكائن المرسل:

```


Sub Main()
    Dim Dtgt As New OneParameter(AddressOf Abs)
    ...
    ...
    MySub(Dtgt)
End Sub

' إجراء يستقبل وسيطة من النوع OneParameter
Sub MySub(ByVal delegatePointer As OneParameter)
    ArabicConsole.WriteLine(delegatePointer.Invoke(-5))
End Sub

```

دعنا نرى الآن ماذا يحدث خلف الكواليس، لأوضح لك حقيقة استخدام الكلمة المحجوزة Delegates كما في السطر الاول من المثال السابق عندما عرفت النوع OneParameter:

```
Delegate Function OneParameter(ByVal X As Integer) As Integer
```

تقنيا، قام مترجم اللغة بتحويل العبارة السابقة إلى فئة باسم OneParameter مشتقة وراثيا من الفئة System.MulticastDelegate (والتي هي أيضا مشتقة من الفئة System.Delegate)، أي يمكنك افتراض أن الشيفرة السابقة قد تم كتابتها لتعريف فئة OneParameter:

```

هذا مجرد افتراض، فلا تطبقه في الواقع '
Class OneParameter
    Inherits System.MulticastDelegate
    ...
    ...
End Class

```

لذلك، عندما أردنا إنشاء نسخة جديدة من الكائن اضطررنا لاستخدام الكلمة المحجوزة New ومن ثم إرسال القيم لوسيطات المشيد:

```
Dim Dtgt As New OneParameter(AddressOf Abs)
```

ما أود الوصول إليه هو ان DlgT مؤشر لكائن يحتوي على خصائص وطرق إضافية (ذكرت إحداها وهي الطريقة Invoke() والتي تستدعي الإجراء المشار إليه)، كما يمكنك التعامل مع هذا الكائن كما تتعامل من كائنات الفئات التي تعرفها بنفسك لتتمكن من كتابة شيئاً مثل:

```
Dim DlgT As New OneParameter(AddressOf Abs)
Dim X As OneParameter

!سناد قيم بين كائنين '
X = DlgT

ArabicConsole.WriteLine(X.Invoke(-5)) ' 5
```

إجراءات الفئات

في الفقرة السابقة طبقنا التفويض على الإجراءات الستاتيكية، اما إن رغبت في تطبيق التفويض على إجراءات الفئات فالعملية ستنتم بنفس الأسلوب السابق إلا انك بحاجة إلى استخدام اسم الكائن الذي يتبع له الإجراء عند إرسال عنوان الإجراء (باستخدام AddressOf) إلى المؤشر المفوض:

```
Class TestClass
    Sub TestMethod()
        ArabicConsole.WriteLine("طريقة من الكائن")
    End Sub
End Class

Module Module1
    Delegate Sub NoParameter()

    Sub Main()
        Dim testObject As New TestClass()

        ! لاحظ هنا استخدام اسم الكائن testObject
        Dim dlgT As New NoParameter(AddressOf testObject.TestMethod)

        dlgT.Invoke()
    End Sub
End Module
```

باستخدام التفويض، يمكنك تخطي حاجز المنع من استدعاء إجراءات الفئة من داخل الإجراءات المشتركة في الفئة، حيث وكما ذكرت في الفصل الثالث **الفئات والكائنات** أنه لا يمكنك استدعاء طريقة في الفئة من داخل طريقة مشتركة في نفس الفئة:

```

Class TestClass
    Sub InstanceMethod()
        ...
        ...
    End Sub

    Shared Sub SharedMethod()
        ' لا يمكنك عمل ذلك
        InstanceMethod()
    End Sub
End Class

```

تستطيع تطبيق العملية السابقة باستخدام التفويض بعشرات الأساليب إما باستخدام متغيرات عامة Global Variables أو إرسالها كوسيطات للإجراءات المشتركة:

```

Class TestClass
    Sub InstanceMethod()
        ArabicConsole.WriteLine("تم استدعاء الطريقة")
    End Sub

    Shared Sub SharedMethod(ByVal methodPointer As NoParameter)
        methodPointer.Invoke()
    End Sub
End Class

Module Module1
    Delegate Sub NoParameter()

    Sub Main()
        Dim testObject As New TestClass()
        Dim dlgt As New NoParameter(AddressOf
testObject.InstanceMethod)

        ' InstanceMethod() سيتم استدعاء الطريقة
        ' SharedMethod() من خلال الطريقة المشتركة
        testObject.SharedMethod(dlgt)
    End Sub
End Module

```

محاكاة الأحداث

إن استخدمت المنطق البرمجي قليلا، فسرعان ما ستكتشف ان الأحداث Events التي تعرفها في الفئات تطبق أسلوب التفويض Delegates ولكن بصيغة مختلفة، فالحدث Die التالي:

```

Class Person
    Event Die()

    Sub Kill()
        RaiseEvent Die()
    End Sub
End Class

```

```
Module Module1
    Sub Main()
        Dim Turki As New Person()

        AddHandler Turki.Die, AddressOf PeronHasDied

        Turki.Kill()
    End Sub

    Sub PeronHasDied()
        ArabicConsole.WriteLine("لقد توفي الشخص")
    End Sub
End Module
```

ما هو إلا تفويض متمثل في الاسم Die، فأنت تقوم بتحديد عنوان الإجراء الذي سيتم تنفيذه باستخدام AddHandler. يمكنك أيضا الاستغناء عن صيغة إطلاق الأحداث السابق وتعريف تفويض (باسم NoParameter مثلا) ومن ثم التصريح عن متغير في الفئة يحمل اسم الحدث Die، وان أردت إطلاق هذا الحدث ستقوم باستدعاء الطريقة Invoke() عوضا عن الأمر RaiseEvent. الشيفرة التالية هي نفس الشيفرة السابقة، ولكني هنا استخدمت التفويض لإطلاق الحدث بدلاً من الصيغة المباشرة:

```
Delegate Sub NoParameter()

Class Person
    Public Die As NoParameter

    Sub Kill()
        Die.invoke()
    End Sub
End Class

Module Module1
    Sub Main()
        Dim Turki As New Person()

        Turki.Die = New NoParameter(AddressOf PeronHasDied)

        Turki.Kill()
    End Sub

    Sub PeronHasDied()
        ArabicConsole.WriteLine("لقد توفي الشخص")
    End Sub
End Module
```

دمج التفويضات

من الأشياء الرائعة جدا في استخدام تقنية التفويض، هو إمكانية دمج أكثر من تفويض في مؤشر واحد بحيث يمكنك تنفيذ أكثر من إجراء باستدعاء واحد لتضرب أكثر من عصفور بحجر واحد. فمثلا، لنفترض ان لدينا الاجرائين التاليين:

```
Sub MySub1()  
    ArabicConsole.WriteLine("الإجراء الاول")  
End Sub  
  
Sub MySub2()  
    ArabicConsole.WriteLine("الإجراء الثاني")  
End Sub
```

ونريد استدعاء الاجرائين باستخدام التفويض، فإنه يتوجب علينا استدعاء كل إجراء على حدة - باستخدام الطريقة `Invoke()`:

```
Delegate Sub NoParameter()  
  
sub Main ()  
    Dim dlgt As New NoParameter(AddressOf MySub1)  
    Dim dlgt2 As New NoParameter(AddressOf MySub2)  
  
    dlgt.Invoke()  
    dlgt2.Invoke()  
End Sub
```

اما عند الدمج فلست بحاجة لكل هذا، حيث يكفي استخدام الطريقة `Combine()` لندمج اكثر من تفويض في مؤشر واحد، لنتمكن من تنفيذ الاجرائين `MySub1()` و `MySub2()` باستدعاء واحد فقط للطريقة `Invoke()`. نستطيع فعل ذلك بإنشاء متغير ثالث للقيام بهذه العملية:

```
Delegate Sub NoParameter()  
  
Sub Main()  
    Dim dlgt As New NoParameter(AddressOf MySub1)  
    Dim dlgt2 As New NoParameter(AddressOf MySub2)  
    Dim dlgt3 As NoParameter  
  
    dlgt3 = dlgt.Combine(dlgt, dlgt2)  
  
    ' سيتم تنفيذ كلا الاجرائين MySub1 و MySub2  
    dlgt3.Invoke()  
End Sub
```

أو استخدام متغير واحد فقط لتنفيذ هذه العملية، بحيث يتم إنشاء الكائنات لحظة إرسال الوسيطات:


```
Delegate Sub NoParameter()

Sub Main()
    Dim dlgt As NoParameter

    dlgt = dlgt.Combine(New NoParameter(AddressOf MySub1), _
        New NoParameter(AddressOf MySub2))

    ' MySub2 و MySub1 سيتم تنفيذ كلا الاجرائين
    dlgt.Invoke()
End Sub
```

الطريقة Combine() طريقة مشتركة Shared Method لذلك تمكنا من استدعائها قبل إنشاء نسخة جديدة من الكائن في المؤشر dlgt السابق. ودعني هنا أنبهك إلى أن هذه الطريقة ستعود بقيمة من النوع System.Delegate لذلك لابد من استخدام المعامل CType ان كانت العبارة Option Strict On مفعلة:

```

dlgt = CType(dlgt.Combine( New NoParameter(AddressOf MySub1), _
    New NoParameter(AddressOf MySub2) ), NoParameter)
```

المزيد أيضا، يمكن دمج إجراء ثالث في المؤشر dlgt السابق، وعند استدعاء الطريقة Invoke() سيتم تنفيذ جميع الإجراءات الثلاثة بالتتالي (الأقدم فالأحدث):

```
' دمج إجراء ثالث
dlgt = CType(dlgt.Combine(dlgt, New NoParameter(AddressOf MySub3)), _
    NoParameter)

' سيتم تنفيذ الإجراءات
' MySub1()
' MySub2()
' MySub3()
' بالتسلسل
dlgt.Invoke()
```

نقطة أخرى حول دمج التفويضات، ففي حالة ما اذا كان نوع الإجراء المفوض Function فعليك معرفة أن الإجراء الأخير هو الذي سيعود بالقيمة:



```
Module Module1
    Delegate Function NoParameterFun() As Boolean

    Sub Main()
        Dim dlgt As New NoParameterFun(AddressOf MyFunction1)
        Dim dlgt2 As New NoParameterFun(AddressOf MyFunction2)

        dlgt = CType(dlgt.Combine(dlgt, dlgt2), NoParameterFun)

        ArabicConsole.WriteLine(dlgt.Invoke()) ' True
    End Sub

    Function MyFunction1() As Boolean
        Return False
    End Function

    Function MyFunction2() As Boolean
        Return True
    End Function
End Module
```

أخيراً، إن رغبت في حذف الإجراء من مؤشر التفويض، فلن تجد أسهل ولا أجمل من الطريقة `Remove()`:



```
dlgt = CType(dlgt.Remove(dlgt, dlgt2), NoParameterFun)

ArabicConsole.WriteLine(dlgt.Invoke()) ' False
```

ملاحظة

الطريقة `Remove()` السابقة تحذف آخر إجراء تم إضافته للمتغير المفوض. مع ذلك، جميع الإجراءات المدمجة في المتغير تحفظ في الخاصية `GetInvocationList` وهي مصفوفة. لذلك، الشيفرة التالية تمكنك من حذف الإجراء الأول وليس الأخير:

```
dlgt.Remove(dlgt, dlgt.GetInvocationList(0))
ArabicConsole.WriteLine(dlgt.Invoke()) ' True
```

المواصفات

مصطلح **المواصفات Attributes** التي يوفرها لك إطار عمل .NET Framework جديدة - نسبيا - في لغات البرمجة، الهدف منها هو ان بعض محتويات مشاريعك قد لا تعتمد على كتابة الشيفرات المصدرية Source Codes بشكل كامل، وإنما قد تشمل تعاريف ونصوص حرفية Textual وعبارات توجيه للمترجم Compiler.

فمثلا، المعلومات العامة حول الفئات وأعضائها تعتمد على النصوص لتشرح الغرض منها ومتطلباتها أو حتى الوصول إلى الصفحة المناسبة لملف التعليمات الخاص بها. هناك أيضا بعض اوامر التوجيه الخاصة بالمترجم Compiler والتي تمكنك من التحكم في الشيفرات التي ترغب في ترجمتها ان تحققت شروط معينة. المزيد أيضا، قد تفرض وتحدد مواقع معينة لمتغيرات التركيبات من نوع Structure وطريقة تحديد مواقعها بالذاكرة.

بعض هذه الأمور متوفرة في لغات البرمجة ولكن لكل لغة طريقة خاصة ومعينة في تطبيقها، فنجد في لغة C/C++ العبارات #pragma للسيطرة على المترجم، واما في لغة Visual Basic 6 فقد كان يعتمد بشكل كبير على صناديق الحوار التي تنشئ ملفات خاصة بها (ك-ctl. و idl). لحفظ هذه المواصفات. ولكن مع إطار عمل .NET Framework، فقد توحدت فكرة تطبيق المواصفات Attributes بين لغات .NET الأخرى. ليس هذا فقط، بل أصبحت تمكنك من بناء مواصفات خاصة بك.

صيغة كتابة المواصفات في Visual Basic .NET

في البداية عليك معرفة ان المواصفات ما هي الا فئات Classes ولكن لها صيغة قياسية لا بد من إتباعها لتتمكن من استخدامها في مشاريعك، استخدم علامة الأقواس المعقوفة < و > لكتابة المواصفة:

```
<System.ComponentModel.DefaultProperty ("XXX")>
```

وبدلا من كتابة الاسم الكامل للمواصفة كما في المثال السابق، اقترح عليك باستيراد مجال الأسماء Namespace الذي توجد فيه المواصفة باستخدام Imports، لتختصر على نفسك كتابة الاسم الكامل في كل مرة تود فيها استخدام المواصفة:

```
Imports System.ComponentModel
<DefaultProperty ("XXX")>
```

السؤال هنا أين ستكتب الموصفة؟ والجواب يعتمد على نوع الموصفة التي ستستخدمها، فبعض الموصفات يمكن كتابتها في أي جزء أو كتلة من شيفراتك المصدريّة، والبعض الآخر محصور في أماكن معينة كما ستري لاحقاً. فمثلاً، الموصفة DebuggerStepThrough لن تستطيع استخدامها إلا عند تعريف الإجراءات فقط:

```
<System.Diagnostics.DebuggerStepThrough()> Sub MySub()  
...  
...  
...  
End Sub
```

قياسياً سنستخدم المعامل _ لفصل الموصفة عن الشيفرة المصدريّة بحيث نتوزع على سطرين مختلفين لتسهيل قراءة وتتبع الشيفرة:

```
<System.Diagnostics.DebuggerStepThrough()> _  
Sub MySub()  
...  
...  
...  
End Sub
```

موصفات من إطار عمل .NET Framework.

يوفر لك إطار عمل .NET Framework العشرات من الموصفات والتي يمكن استخدامها في أجزاء متفرقة من البرنامج لأداء وظائف معينة، في الفقرات التالية سنلقي الضوء على بعض هذه الموصفات والموجهة للغة البرمجة .NET Visual Basic، كما سأخصص فقرة كاملة لأعرض لك كيفية بناء موصفة خاصة بك.

الموصفة Conditional Attribute

عملية توجيه المترجم تقتضي ترجمة سطور معينة من الشيفرة المصدريّة لحظة تحقق شرط معين، في معظم لغات البرمجة السابقة يستخدم الموجه #If لهذا الغرض:

```
#If MyCondition Then  
Sub MySub()  
...  
...  
End Sub  
#End If
```

```
Sub Main
    MySub ()
    ...
    ...
    MySub ()
End Sub
```

المشكلة في هذا الأسلوب تبرز إن أردت إلغاء الإجراء MySub() السابق (عن طريق تغيير قيمة الشرط MyCondition ليكون False) فستظهر مئات الأخطاء في باقي سطور البرمجة - لحظة الترجمة - عند كل استدعاء للإجراء MySub()، احد الحلول هو استخدام الموجه #If عند كل استدعاء:

```
Sub Main
    #If MyCondition Then
        MySub ()
    #End If
    ...
    ...
    #If MyCondition Then
        MySub ()
    #End If
    ...
    ...
End Sub
```

صحيح ان الاسلوب السابق غير عملي، لذلك قد تدخل عبارة الشرط للموجه داخل الإجراء MySub() وتسهل عليك المهمة:

```
Sub MySub ()
    #If MyCondition Then
        ...
        ...
    #End If
End Sub
```

مع ذلك، إن كانت نتيجة الشرط MyCondition تساوي False فإن مئات السطور من الشيفرات المصدرية (والتي تستدعي الإجراء MySub() بالتحديد) أصبحت غير ذات قيمة، والتي ستتسبب في بطء عملية الترجمة - التنفيذ أيضا - بالإضافة إلى كبر حجم البرنامج.

يوفر لك إطار عمل .NET Framework المواصفة Conditional Attribute والتي يمكنك من توجيه المترجم عند تحقق شرط معين، وطريقة استخدامها يكون عند تعريف الإجراءات من نوع Sub:

```
<Conditional("MyCondition")> _
Sub MySub()
    ...
End Sub

Sub Main
    MySub ()
    ...
    MySub ()
    ...
End Sub
```

الإجراء MySub() السابق وجميع السطور التي ستستدعيه في كافة ملفات المشروع الأخرى، سيتم تجاهلها لحظة الترجمة ولن يتم ترجمتها وتضمينها في الملف النهائي ان كانت نتيجة الشرط MyCondition تساوي صفر أو False.

ملاحظة

لا يمكنك تطبيق المواصفة Conditional على الإجراءات من نوع Function.

بالنسبة للشرط MyCondition السابق فما هو الا ثابت يمكنك تعريفه وتحديد قيمته من صندوق حوار خصائص المشروع Project Property Pages، والانتقال إلى خانة التبويب Configuration Properties ثم خانة التبويب الفرعية Build ثم كتابة اسم الثابت وقيمته تحت خانة النص Custom Constants (شكل 5-2 بأعلى الصفحة المقابلة).



شكل 5-2: تحديد ثوابت الترجمة في صندوق حوار Project Property Pages.

المواصفة DebuggerStepThrough Attributes

في الفصل السابع اكتشاف الأخطاء سأنتقل إلى بعض طرق التنقيح Debugging التي توفرها بيئة Visual Studio .NET. ومنها التنفيذ سطرا سطرا (لعمل ذلك اختر الأمر Step Into من قائمة Debug أو اضغط على المفتاح [F11]). في بعض الإجراءات التي تكون متأكدا من صحتها ولست بحاجة إلى التدقيق فيها سطرا تلو الآخر، يمكنك استخدام المواصفة System.Diagnostics.DebuggerStepThrough عند بداية الإجراء ليتم تنفيذه دفعة واحد:

```
<System.Diagnostics.DebuggerStepThrough(> _
Sub MySub()
    ...
    ...
    ...
End Sub
```

المواصفة Obsolete Attribute

الغرض من هذه المواصفة هو إظهار رسائل التنبيه Warning أو الخطأ لحظة ترجمة البرنامج، يمكنك الاستفادة منها مثلا عندما تقوم بإرسال برنامجك إلى آخرين وتود لفت انتباههم إلى جزء معين من الشيفرة أو التحقق منها:

```
<Obsolete("يا عباس، لا تنسى تغيير شيفرة هذا الإجراء قبل التوزيع") _
Sub MySub()
...
...
End Sub
```

عند الترجمة، ستظهر الرسالة السابقة كإنذار Warning في نافذة المخرجات Output (والتي تصل إليها باختيار الأمر View->Other Windows->Output) (شكل 5-3).



شكل 5-3: رسالة التنبيه Warning ظهرت أثناء الترجمة.

المزيد أيضاً، الموصفة Obsolete تستقبل وسيطة أخرى من النوع Boolean، إن أرسلت لها القيمة True ستحول الرسالة من رسالة تنبيه إلى رسالة خطأ لتوقف عملية الترجمة ولن يتم تنفيذ البرنامج:

```
<Obsolete("الم أقل لك عدل الإجراء يا عنيد", True)> _
Sub MySub()
...
...
End Sub
```

ملاحظة

سواء كانت رسالة تنبيه أو خطأ، فلن يتم عرضها إلا عند السطر الذي يستدعي الإجراء وليس بمجرد استخدام الموصفة Obsolete.

المواصفة StructLayout والمواصفة FieldOffset

استخدام المواصفة StructLayout موجه بشكل حصري إلى التركيبات من النوع Structure، حيث يمكنك من تحديد طريقة ترتيب متغيرات التركيب في الذاكرة، أرسل القيمة LayoutKind.Auto للحصول على أفضل وانسب أداء للمعالج:

```
' استيراد مجال الاسماء لاختصار الشيفرات
Imports System.Runtime.InteropServices

<StructLayout(LayoutKind.Auto)> _
Structure RGBValue
    Dim Red As Byte
    Dim Green As Byte
    Dim Blue As Byte
End Structure
```

أما إن أرسلت القيمة LayoutKind.Explicit فعليك تحديد موقع كل متغير من متغيرات التركيب في الذاكرة بنفسك عن طريقة استخدام المواصفة FieldOffset:

```
<StructLayout(LayoutKind.Explicit)> _
Structure RGBValue
    <FieldOffset(0)> Dim Red As Byte
    <FieldOffset(1)> Dim Green As Byte
    <FieldOffset(2)> Dim Blue As Byte
End Structure
```

استخدامك للمواصفة FieldOffset يعطيك مرونة كبيرة للتحكم في مواقع المتغيرات في الذاكرة، فمثلا التركيب التالي لا يحتجز الا بايت واحد تصل اليه من ثلاث متغيرات:

```
<StructLayout(LayoutKind.Explicit)> _
Structure RGBValue
    <FieldOffset(0)> Dim Red As Byte
    <FieldOffset(0)> Dim Green As Byte
    <FieldOffset(0)> Dim Blue As Byte
End Structure
```

ولتأكد من كلامي:

```
Dim X As RGBValue
X.Blue = 10
ArabicConsole.WriteLine(X.Red) ' 10
```

المزيد أيضاً، تستطيع الوصول إلى قيمة المتغيرات الثلاثة كلها عن طريق متغير واحد، راقب هذا التركيب:

```
<StructLayout(LayoutKind.Explicit)> _
Structure RGBValue
    <FieldOffset(0)> Dim Red As Byte
    <FieldOffset(1)> Dim Green As Byte
    <FieldOffset(2)> Dim Blue As Byte
    <FieldOffset(3)> Dim Extra As Byte
    <FieldOffset(0)> Dim Value As Integer
End Structure
```

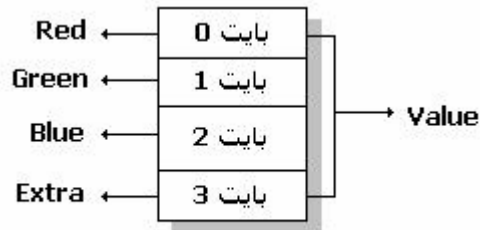
لديك الآن الحرية المطلقة في الوصول إلى قيمة التركيب إما عن طريق المتغير Value (وهو يشمل المتغيرات الأربعة) أو عن طريق كل متغير على حدة:

```
Dim X As RGBValue

X.Value = 0
ArabicConsole.WriteLine(X.Red)           ' 0
ArabicConsole.WriteLine(X.Green)         ' 0
ArabicConsole.WriteLine(X.Blue)          ' 0

X.Red = 255
X.Green = 255
X.Blue = 255
ArabicConsole.WriteLine(X.Value)         ' 16777215
```

(الشكل 5-4) يوضح لك مواقع متغيرات التركيب RGBValue السابق في الذاكرة.



شكل 5-4: مواقع متغيرات التركيب RGBValue في الذاكرة.

بناء مواصفات خاصة

المواصفات الخاصة **Custom Attributes** ما هي إلا فئات **Classes** تقليدية تحتوي على جميع الشيفرات التي تود من المواصفة القيام بها، مع ذلك لابد للفئات -التي ترغب في ان تكون مواصفة **Attributes** - ان تستوفي عدة شروط، سأذكرها تباعاً، وسنطبقها معاً على مراحل ليسهل عليك استيعاب الفكرة.

سأقوم معك ببناء مواصفة **CodeInfo** تمكّنك من الاحتفاظ بمجموعة بيانات تتعلق بالشفرة المصدرية (كاسم كاتبها، اسم مراجعها، تاريخ كتابتها، ... الخ).

الشرط الأول: يجب أن ينتهي اسم الفئة بالكلمة **Attribute**:

```
Class CodeInfoAttribute
End Class
```

الشرط الثاني: ان تكون الفئة مشتقة من الفئة **System.Attribute**:

```
Class CodeInfoAttribute
    Inherits System.Attribute
End Class
```

الشرط الثالث: ضرورة استخدام المواصفة **AttributeUsage** وتحديد المكان الذي يمكن استخدام المواصفة الحالية في شيفرات البرنامج:

```
<AttributeUsage(AttributeTargets.All) _
Class CodeInfoAttribute
    Inherits System.Attribute
End Class
```

تلاحظ اني ارسلت القيمة **AttributeTargets.All** وذلك لأجعل المواصفة الحالية قابلة للاستخدام في جميع أماكن الشيفرات المصدرية، يمكنك إرسال قيم أخرى لحصر مجال استخدام هذه المواصفة في موقع معين، الجدول الموجود في الصفحة التالية يوضح لك قيمة المواصفة، ومجال عملها.

القيمة	المكان الذي ستستخدم فيه الموصوفة الحالية
AttributeTargets.All	في كل الاماكن.
AttributeTargets.Assembly	في المجمع الحالي.
AttributeTargets.Class	عند تعريف الفئات.
AttributeTargets.Constructor	مشيدات الفئات.
AttributeTargets.Delegate	الاجرائات المفوضة - باستخدام الكلمة المحجوزة Delegate.
AttributeTargets.Enum	التركيبات من النوع Enum.
AttributeTargets.Event	الأحداث.
AttributeTargets.Field	الحقول.
AttributeTargets.Interface	الواجهات.
AttributeTargets.Method	الطرق.
AttributeTargets.Module	الوحدات البرمجية.
AttributeTargets.Parameter	وسيطات الإجراءات.
AttributeTargets.Property	الخصائص.
AttributeTargets.Return Value	الاجرائات التي تعود بقيمة Functions.
AttributeTargets.Struct	التركيبات من النوع Structure.

ملاحظة

يمكنك تحديد أكثر من موقع باستخدام المعامل Or لحظة ارسال القيمة للموصوفة AttributeUsage، فالقيمة التالية تحصر مجال استخدام الموصوفة الحالية على الطرق والخصائص فقط:

```
<AttributeUsage(AttributeTargets.Method Or
AttributeTargets.Property)
```

الشرط الرابع: أنواع البيانات المسموح بها في هذا النوع من الفئات هي: Boolean، Byte، Short، Integer، Long، Char، String، Single، Double، Object، التركيب من نوع Enum، المصفوفات أحادية البعد One-Dimensional Array فقط.

سنقوم الآن بتعريف أعضاء الفئة مع الالتزام بالأنواع المحددة في الشرط الرابع:



```
<AttributeUsage(AttributeTargets.All) _
Class CodeInfoAttribute
    Inherits System.Attribute

    Public ProgrammerName As String ' اسم المبرمج
    Public TesterName As String ' اسم المراجع
    Public Tested As Boolean ' تمت المراجعة؟

    Sub New(ByVal programmerName As String)
        Me.ProgrammerName = programmerName
    End Sub
End Class
```

مبروك! المواصفة جاهزة للاستخدام الآن، أرسل وسيطة المشيد لحظة استخدامها في أي مكان من البرنامج (لاحظ تجاهل كتابة الكلمة Attribute عند استدعاء المواصفة):

```
<CodeInfo("تركبي العسيري") > _
Sub DoSomething()
    ...
    ...
End Sub
```

بالنسبة للخصائص الأخرى TesterName و Tested، فيمكنك إسناد القيم لها بهذه الصيغة:



```
<CodeInfo("تركبي العسيري") > _
Class Person

    <CodeInfo("تركبي العسيري", TesterName:="عباس السريع", Tested:=False)> _
    Sub Move ()
        ...
        ...
    End Sub

    <CodeInfo("تركبي العسيري", TesterName:="عبود اللوح", Tested:=True)> _
    Sub Kill ()
        ...
        ...
    End Sub
    ...
    ...
End Class
```

كنوع من الاقتراح، قد تضيف شيفرات إضافية لحفظ بيانات المواصفة CodeInfo في ملفات نصية خارجية.

بهذا أكون قد انتهيت من تشييد بنية أساسية لتكون مبرمج .NET Visual Basic حقيقي بعدما تطرقت إلى جميع المبادئ والأساسيات التي لابد على كل مبرمج Visual Basic .NET من معرفتها وإقناعها للإبحار في برمجة واستخدام مكتبة فئات إطار عمل .NET Framework Class Library والآن فإن لديك مطلق الحرية في الانتقال إلى أي جزء ترغب به، رغم أنني أنصحك بشدة بالالتزام بالتسلسل الموضوع في هذا الكتاب والبدء من الفصل التالي **الفئات الأساسية** حتى تتعرف على اكبر قدر من الفئات الأولية قبل تطوير التطبيقات المتقدمة

الجزء الثاني

إطار عمل .NET Framework

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 بـ Visual Basic .NET. ولكن الإلمام بجميع هذه الفئات يحتاج إلى وقت طويل جداً من عمرك البرمجي، ولا أنصح سراً إن أخبرتك أنني حتى لحظة كتابة الصفحة الأخيرة من هذا الكتاب مازلت أكتشف المزيد والمزيد من هذه الفئات وأعضائها.

يقدم لك هذا الفصل مدخلاً إلى الفئات الأساسية لإطار عمل .NET Framework، وسنستمر في التوغل في معظم باقي الفئات الأخرى بالتدرج في الفصول اللاحقة حتى نهاية هذا الكتاب، وعلي أن أذكرك هنا بأن هذا الكتاب ليس مرجع من مراجع MSDN، فلا تتوقع مني القيام بشرح جميع مكتبة فئات .NET Framework وأعضائها بنفسني. لذلك، عليك البحث عن كافة تفاصيل الفئات وأعضائها في مستندات .NET Documentation.

الفئة System.Object

جميع فئات إطار عمل .NET Framework وجميع التركيبات والأنواع الأخرى مشتقة وراثياً - بشكل مباشر أو غير مباشر - من الفئة System.Object، مما يعني أن جميع الأعضاء المعرفة في الفئة System.Object ستكون موجودة أيضاً في أنواع البيانات التي تنشئها. المزيد أيضاً، يمكنك إسناد أي قيمة إلى المؤشرات من النوع Object (فهي الفئة القاعدية Base Class لذلك تستطيع إسناد أي قيمة من فئة مشتقة Derived Class إلى فئة قاعدية كما وضحت لك في الفصل الرابع الوراثة):

```
' يمكنك استخدام الصيغة المختصرة
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 بإعادة قيادة `Overrides` هذه الطريقة (وهذا ما ستقوم به أنت 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`.

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، إلا أنها 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 تلقائياً 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 أيضاً ولن تتمكن من استدعائها إلا من داخل الفئة -حالتها كحال الطريقة السابقة MemberwiseClone(). تحدثت عن هذه الطريقة سابقاً في الفصل الثالث الفئات والكائنات، وذكرت أنها تعمل عمل المهدمات Destructors.

البيانات المرجعية والبيانات ذات القيمة مرة أخرى

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

الأعداد Numbers (سواء كانت صحيحة أو عشرية)، القيم المنطقية Booleans، التركيبات من النوع Structures أو Enums جميعها بيانات من النوع ذو القيمة Value Type، وعندما اطلق عليها كلمة فئات عديدة أو فئات من النوع Boolean -في هذا الفصل في التحديد- فلا اعني انها فئات حقيقية كما تعرفها بداخل التركيب Class ... End Class، حيث أن 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 وراثياً وبشكل تلقائي -من الفئة System.ValueType، حيث أنها تحتوي على طرق متشابهة في الأداء (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 أو جميع البيانات العددية Numbers، التركيبات من النوع Structures أو Enums جميعها من الأنواع ذات القيمة Value Type. المزيد أيضاً، يمكنك معرفة ما إذا كانت فئة معينة من النوع المرجعي Reference Type أو ذو القيمة Value Type بقراءة مستندات .NET Documentation أو 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 وأقل استهلاكاً للذاكرة لسببين: الأول أن بيانات المتغيرات ذات القيمة تصل إليها مباشرة عن طريق المتغير نفسه بينما بيانات المتغيرات المرجعية فالوصول إليها يكون بالمرور أولاً بالمؤشر (المتمثل في المتغير نفسه) لتصل إلى بيانات الكائن الموجود في القسم Managed Heap من الذاكرة. السبب الثاني أن عملية تفرغ محتويات البيانات من الذاكرة لا تتطلب استخدام الـ Garbage Collection فهي مخصصة للقسم Managed Heap أي للبيانات المرجعية. لذلك، يفضل دائماً استخدام المتغيرات ذات القيمة Value Type في برامجك، والطريقة الوحيدة التي يمكنك 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، مثلاً عند عملية إسناد القيمة، فلو كان لدينا التركيب التالي:

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

فإن 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 أن عملية إسناد القيم بين المتغيرات ذات القيمة تؤدي إلى نسخ جميع محتويات وعناصر التركيب التابع له المتغير، بينما نجد في المتغيرات المرجعية أن إسناد القيم يؤدي إلى نسخ المؤشرات فقط (حجمها لا يتجاوز 4 بايت).

الصندوقية واللاصندوقية

حالة أخرى تجعل المتغيرات المرجعية أسرع من المتغيرات ذات القيمة تعرف **بالصندوقية Boxing** و**اللاصندوقية Unboxing**. تحدث الصندوقية نتيجة إسناد قيمة لمتغير من نوع ذو قيمة Value Type إلى متغير من النوع Object، أو إرسال متغير من النوع ذو القيمة إلى إجراء يستقبل وسيطة من النوع Object:

```
Dim X As Integer = 10

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

عملية الصندوقية 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. وإن قمت بتطبيق العكس (إسناد قيمة متغير من النوع Object إلى متغير ذات قيمة Value Type)، فإنك ستحدث ما يعرف باللاصندوقية Unboxing:

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

الصندوقية واللاصندوقية تستهلك وقت إضافي في عملية إسناد القيم، أو حتى إرسال القيم إلى وسيطات من النوع Object إلى الإجراءات (كالطريقة WriteLine() التابع للكائن ArabicConsole). خلاصة القول، استخدم البيانات ذات القيمة Value Type دائماً إن كنت لا ترغب في تطوير النوع باستخدام الوراثة ولا تتوقع عمليات إسناد قيم متعددة أو حدوث أي عمليات صندوقية أو لاصندوقية أخرى.

الفتات 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)
```

وذلك لأننا 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:

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

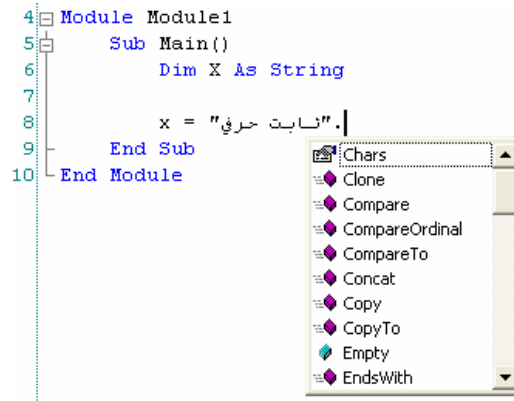
Dim Y As String = "AAAAA"
```

اعلم جيداً 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 إيجاد الثوائب في برنامجك. وبما أن المتغيرات الحرفية من النوع المرجعي Reference Type، فإن 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 إرباكاً أكثر إن زدتك من الشعر بيت وأخبرت أنك أنه حتى الثوابت الحرفية (التي تكتبها داخل علامات التنصيص المزدوجة "و") يتعامل معها Visual Basic .NET على أنها 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 تتسنى ان الثوابت الحرفية (والتي هي كائنات) تحفظ في القسم Managed Heap من الذاكرة. دعني اضيف شيء اخر مهم حول الثوابت الحرفية، مترجم اللغة .NET Visual Basic 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 بواحد فقط، ففي السطر التالي:

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

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

الخصائص والطرق

لا تحتوي الفئات من النوع String إلا على خاصيتين 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-ج-د"
Dim y As String = "A-B-C-D"

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

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 شرحاً مفصلاً 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 أبداً، 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:

```
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 أو 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 معيناً 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
```

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 ب 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 استناداً إلى نوع القيمة التي تحددها كوسيلة ترسلها لهذه الطريقة، ففي الشيفرة التالية سيتم فصل كلمات الجملة وذلك لأنني أرسلت الحرف "c" مع الوسيلة:

```
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

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

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

أخيراً، 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 الوسيطين 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 الانجليزية Case-sensitive، حيث ان "turki" لا تساوي "TURKI" عند التعامل مع الطريقة 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 (لاحقاً سنتعلم كيف يمكنك تجاهلها) بينما علامة الكشيدة سيتم تجاهلها حيث أن "محمد" تساوي "محمّد":

```
ArabicConsole.WriteLine (String.Compare ("كلمة", "كَلِمَة")) ' 1
ArabicConsole.WriteLine(String.Compare("محمد", "محمّد")) ' 0
```

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", "محمد")) ' -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**.

تمتلك 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

المشيد، وإن كنت لا تعلم الاسم أو 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 لجهازك 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** أو رقم المعرف **1025**، لذلك أستطيع إنشاء كائن يحمل اعدادات المملكة العربية السعودية على الفور:

```
Dim KSA As New CultureInfo("ar-SA")
' أو
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	الأردن.
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	الإمارات.
ar-YE	0x2401	اليمن.
ar	0x0001	باقي الدول العربية.

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

```
0
ArabicConsole.WriteLine(String.Compare("محمد", "عبد",
, 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 تتمكنك من تجاهل علامات التشكيل وذلك بإرسال القيمة `CompareOptions.IgnoreSymbols` لها:

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

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

' 0
ArabicConsole.WriteLine(KSA.CompareInfo.Compare("كلمة", "كَلِمَة", _
    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", "أحمد", _
    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 المرسله، 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 جداً 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، فالكثيدة سيتم تجاهلها بحسب الإعدادات الإقليمية للنظام الحالي، بينما سيتم أخذ علامات التشكيل بعين الاعتبار:

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

ArabicConsole.WriteLine(x.IndexOf("الكلمة")) ' 39
ArabicConsole.WriteLine(x.IndexOf("الكلمة")) ' 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 الكلمة أو 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 أن 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 يمكنك من تحديد نوع الحرف (بالصيغة (Isxxx()، وتعود بقيمة منطقية من النوع Boolean تمثل نتيجة الاختبار، اذكر لك بعضها منها:

```
' هل الحرف المرسل رقم '
ArabicConsole.WriteLine(Char.IsDigit("1" c)) ' True
' هل الحرف المرسل حرف اعجمي '
ArabicConsole.WriteLine(Char.IsLetter("ت" c)) ' True
' هل الحرف المرسل رقم او حرف '
ArabicConsole.WriteLine(Char.IsLetterOrDigit("X" c)) ' True
' هل الحرف المرسل حرف اعجمي 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 المرسل:

```
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
```

قامت بإنشاء ثلاثة كائنات حرفية في ذاكرة Managed Heap هي "برعي ابو" و "جبهة" 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 الكائن مما يؤدي إلى بطء في التعامل مع البيانات الحرفية بصفة عامة والحروف الطويلة بصفة خاصة.

انظر أيضا

كان لي حديث مطول عن المجموعة Garbage Collection وكيفية تحرير الذاكرة من البيانات المرجعية Reference Type سابقا في الفصل الثالث الفئات والكائنات.

من هنا يأتي دور الفئة System.Text.StringBuilder، حيث تمكنك هذه الفئة من احتجاز مساحة ثابتة للقيم الحرفية (تسمى **String Buffer**) لنتم عملية تعديل قيم ومحتوى البيانات الحرفية في نفس المكان دون الحاجة إلى إنشاء نسخة جديدة من الكائن الحرفي. يمكنك تحديد عدد الحروف في الـ String Buffer عن طريق الخاصية Capacity او لحظة إنشاء الكائن وإرسال القيمة إلى وسيطة المشيد:

```
' حجم الـ String Buffer 100 بايت
' فهو 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)
```

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 -خلافًا 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 مثلاً إسناد قيمة ثابت حرفي إلى متغير من النوع StringBuilder:

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

Y = "رسالة خطأ" ' عباس السريع
```

الفئات العددية

الفئات العددية هي مجموعة من الأنواع المعرفة في إطار عمل .NET Framework. تختلف في نوع وحجم القيمة العددية التي تسندها إليها وهي: Byte، Short، Integer، Long، Single، Double، و Decimal هذا النوع من البيانات 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 دائماً في الاعتبار لحظة استخدامها.

انظر أيضا

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

الخصائص والطرق

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

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 كوسيط مع الطريقة ToString()، كالرمز # الذي يمثل عدداً أو مسافة خالية، أو رمز الصفر 0 الذي يمثل عدداً أو 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 شرحاً وافياً لها في مستندات .NET Documentation. كالرمز "." الذي يمثل الفاصلة العشرية، أو الرمز "," الذي يمثل فاصلة الآلاف، أو الرمز "%" الذي يؤدي إلى عرض النسبة المئوية، أو 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 لجهازك الشخصي وهي العربية (المملكة العربية السعودية):

```
Dim MyDouble As Double = 123456789
' عملة Currency
ArabicConsole.WriteLine(MyDouble.ToString("C")) ' ١٢٣,٤٥٦,٧٨٩.٠٠ ر.س.
' علمي Scientific
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` كوسيلة ثانية للطريقة `ToString()`، المخرجات التالية من البيئة الألمانية (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 أيضاً، 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 تتسبب 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()` للأس، وطريقة اللوغارثم `Log()` أو اللوغارثم العشري `Log10()`، والطريقة `Sign()` التي تعود بالقيمة 1 إن كان العدد موجب أو -1 إن كان سالب أو 0 إن كان صفراً:

```
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، وإن 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 الأعداد بالطريقة `IEEERemainder()` التي تعود بباقي القسمة، والطريقة `Ceiling()` التي تحذف الفاصلة العشرية وتعود بقيمة صحيحة أكبر من أو تساوي العدد المرسل، بينما الطريقة `Floor()` تعود بقيمة صحيحة أصغر من أو 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 أكثر 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
```

أخيراً، طرق الدوال المثلثية Sin()، Cos()، Tan() ... الخ تعود بالقيمة المناسبة استناداً
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، فمثلاً
الطريقة Next() تعود بقيمة عشوائية صحيحة Integer موجبة:

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

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

أما 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 أيضاً، يمكنك تخصيص مجال معين للأعداد العشوائية بإرسال أصغر قيمة وأكبر
قيمة كوسيطات مع الطريقتين السابقتين:

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

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

اخيرا، يمكنك ملء مصفوفة عديدة 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 أساليب: كإرسال 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 صباحاً، ويمكن تحديد الوقت بإرسال الساعة، الدقيقة، والثانية على التوالي:

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

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

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

أو إرسال ثابت الوقت والتاريخ بين علامتي # و #:

```
' 20 فبراير لعام 2003
' الساعة 2:00:00 صباحاً
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، أو ثانية وهي: AddYears()، AddMonths()، AddDays()، AddHours()، AddMinutes()، و AddSeconds():

```
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)
```

أخيراً، 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 الإقليمية الحالية في النظام.

يمكنك تحويل المتغير التاريخي إلى متغير يعتمد التقويم الهجري بإرسال 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 أيضاً- 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 تعتمد أجهزة الحاسب (فقد يكون الشهر 29 او 30 يوماً). لذلك، عليك إعلام المستخدم بأنه يتوجب عليه إجراء التعديلات اللازمة للتقويم الهجري من لوحة التحكم (شكل 6-2 بالصفحة المقابلة)، أو 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
```



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 يمكنك 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
```

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/dd"))
' 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 أو 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()

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

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

الفئات من النوع Enum

جميع التركيبات من النوع Enum التي تعرفها في شيفراتك المصدريّة، 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 عوضاً 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 يمكنك من إسناد قيمة للمتغير بكتابة اسم العضو حرفياً:

```
' 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`، أما إن 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 عديدة، 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
```

اخيراً، 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
```

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 And, Or, XOR... الخ. لعمل ذلك، أضف الموصافة 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 الموصافة، 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 بأن جميع المصفوفات من النوع المرجعي Reference Type حتى وإن 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)
```

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

اما ان اردت نسخ قيم المصفوفات وليس مؤشراتهما، 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
' المصفوفة X()
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 GetLength 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 بأن عملية المسح على عناصر المصفوفة ستتم بشكل أفقي وليس عمودي (أي أن البداية ستكون من

العنصر (0, 0) فالعنصر (0, 1) فالعنصر (0, x) فالعنصر (1, 0) فالعنصر (1, 1)
وهكذا:

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

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 نجد 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)
...
...
```

من الأشياء الطريفة 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 بأن هذه الطريقة تعمل بنفس الكفاءة مع العناصر الحرفية أيضاً، حيث أنها تقوم بترتيب العناصر استناداً 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 أيضاً إخبارك بأن الطريقة Sort() تتجاهل علامة الكشيدة وعلامات التشكيل بحيث لا تخل بعملية الفرز، فلا تحمل هم فرز الحروف العربية.

ملاحظة

رغم أن عملية الفرز تتم استناداً إلى المقابل العددي للحرف الأبجدي، إلا أن الطابع اللغوي يغلب أكثر، حيث يتم تجاهل حالة الأحرف الصغيرة Small والكبيرة Capital للحروف الإنجليزية. لذلك، لا تعتقد أن 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 المرسله 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 = {"خالد", "احمد", "تركي"}

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، ولكنني 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 إليها 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 وراثياً 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()).

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.

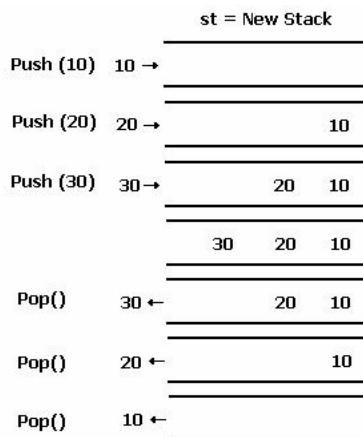
الفئة Stack

تمتلك الفئة Stack من تطبيق خوارزم (LIFO) Last-In-First-Out 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.

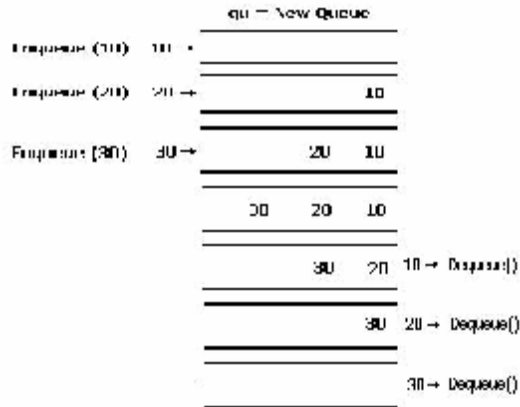
الفئة Queue

تمتلك الفئة Queue من تطبيق خوارزم (FIFO) First-In-First-Out 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 أو 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) ' 100 عنصر وليس 101
```

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

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

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

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

إذا 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()، واللذان تمكنانك من إضافة مجموعة إلى نفس المجموعة، أو 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() أو 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.

اكتشاف الأخطاء

قرأت مرة في احد مجلات البرمجة الجملة التالية: "إن كانت عملية التنقيح Debugging هي الحل لزوال الأخطاء، فإن 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 أحياناً 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 اكتشافاً وتنقيحاً، حيث تظهر لك نافذة محرر الشفرات Code Editor خطأ متعرجاً 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'.
```

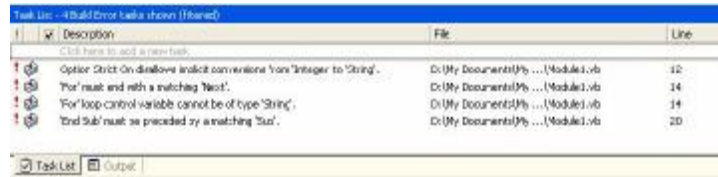
شكل 7-1: ظهور أداة التلميح عند الخط المتعرج لتوضيح خطأ وقت التصميم.

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



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

إن قمت بالضغط على الزر Yes، سيتم تنفيذ آخر نسخة صحيحة (لا توجد بها أخطاء) من البرنامج، أما نترك 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 جداً ولا يحتاج إلى فلسفة، فكل ما هو مطلوب منك الانتقال إلى السطر الذي وقع فيها الخطأ وتصحيحه.

أخطاء وقت التنفيذ

كمبرمج .NET Visual Basic، أنت 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 أغلب الأحوال هو عدم توقع المبرمج بالتغيرات الخارجية لبيئة المنصة التي يعمل بها البرنامج، فمثلاً في الفصل القادم الملفات والمجلدات سترى

كيف يمكنك نقل ملف في الجهاز من مسار إلى آخر 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 آخر لا يحتوي على نسخة من هذا الملف أو نفذته مرة أخرى في جهازك الشخصي - بعد عملية نقل الملف، ستظهر لك بيئة التطوير Visual Studio .NET رسالة استثناء Exception (شكل 7-4) تفيدك بأن الملف D:\File.EXE غير موجود.



شكل 7-4: رسالة الاستثناء Exception.

كما أخبرتك سابقاً بصيغة أخرى - عليك كمبرمج .NET Visual Basic من أخذ الاستثناء دائماً وأبداً 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 أنك تستطيع الالتفاف حول الاستثناءات بهذه السهولة، حيث أن الأمر أعقد من أن يتم حله بجملة شرط واحدة، تخيل مثلاً أن 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 أخرى إن لم تتحقق أيضاً من قابلية الكتابة باستخدام الطريقة GetAttribtues():

```
Imports System.IO

If Not CBool(File.GetAttributes("C:\Test\program.exe") And _
    FileAttributes.ReadOnly) Then
    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 القراءة فقط Read Only ولكنه مفتوح من قبل برنامج اخر وعليه خاصية الإقفال، او ان يكون المسار الهدف غير قابل للكتابة (كأقراص CD-ROM مثلا)، او ان المسار الهدف لا توجد به مساحة كافية لنقل الملف اليه، او... او... وغيرها الكثير من الاحتمالات التي لا يمكنك تداركها بالاعتماد على الجمل الشرطية.

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

الشوائب

حتى لو وصلت إلى مرحلة ما بعد الاحتراف في البرمجة، فانه يستحيل عليك كتابة برنامج لا يحتوي على شوائب **Bugs**. الشوائب -في لغات البرمجة- هي أخطاء غير متوقعة ينتج عنها تصرف غير متوقع او منطقي للبرنامج، فمثلا الاجراء التالي يقوم بإنشاء عدد يمكنك من وقف 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 مشاكل، ولكن يوجد في الخوارزم السابق شائب Bug برمجي خطير، حيث ان الاجراء السابق يعتمد اعتماد كلي على ساعة الجهاز، وان قمت بارسال عدد معين من الثواني بحيث يزيد ويتعدى عن وقت نهاية اليوم (الساعة 23:59:59) سيتم تنفيذ الحلقة إلى مالا نهاية، مما يضطر المستخدم إلى

إيقاف عمل البرنامج باستخدام نافذة Windows Task Manager (التي تصل إليها 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 بأنه 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، فجد مثلا الاستثناءات DivideByZeroException، OverflowException و NotFiniteNumberException والخاصة بالاعداد والعمليات الرياضية، ونجد ايضا الاستثناءات FileNotFoundException، EndOfStreamException، DirectoryNotFoundException... 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 خاصيتين 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.

الشفيرة التي سببت في حدوث خطأ يقال انها ترمي استثناء **Throw an exception**، اما الشفيرة 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 نتوقع حدوث استثناء فيه، يمكنك استخدام التركيب Try ... Catch ... End Try. اصف 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
```

تطبيقا، سنستخدم 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، إذ يفضل استخدام نوع الفئة بكتابة اسمها (راجع مكتبة MSDN لمعرفة أسماء كافة الفئات
الأخرى):

```
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
```

عند وقوع الاستثناء في الشيفرة السابقة، سيتم التحقق من نوع الاستثناء بدءاً من أول عبارة Catch حتى النهاية، وإن لم تتوافق الفئة مع الاستثناء فسيتم الانتقال إلى الاستثناء الأخير Exception بشكل تلقائي، حيث أن الكائن Exception يمثل أي استثناء تم رميه مهما اختلف نوعه. أخيراً، يمكنك الخروج من داخل التركيب Try ... Catch ... End Try في أي وقت بكتابة العبارة Exit Try.

ملاحظة

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

استخدام When:

يمكنك استخدام الكلمة المحجوزة When برفقة العبارة Catch إن أردت إضافة شرط إضافي لعملية قنص الاستثناءات، فبدلاً 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 أجمل وتصميم أرقى 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 أو 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
```

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()
```

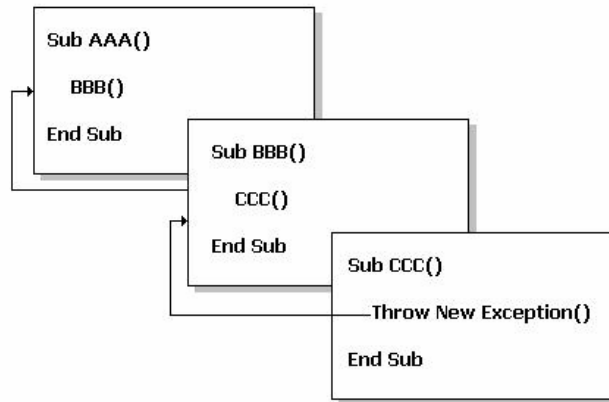
يمكنك تحديد نص الرسالة في الخاصية Message لكائن الاستثناء لحظة الرمي باستخدام الأمر Throw:

```
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، أو 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 نفهم 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
```



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

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

جرب القيام بتدراك الاستثناء في الإجراء الأول `AAA()` ويمكنك معرفة مسار الإجراءات التي يشملها الاستثناء عن طريق الخاصية `StackTrace` - كما لمحت إليها سابقاً:

```

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 الحلوليات وتكثر شيفراتك المصدريه 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 NET Framework. 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

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

والآن يمكنك تقادي 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 .NET Visual Basic عن سائر لغات .NET. الاخرى بوجود الكائن Err الذي يمكنك من تقادي ورمي الاستثناءات. وفي حقيقة الامر، وفرت Microsoft هذا الكائن خصيصا لمبرمجي Visual Basic 6 اعتقادا منهم ان .NET Visual Basic نسخة جديدة من 1->6 Visual Basic (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 تقادي الاستثناءات ورميها ايضا مع الكائن Err ولكن بصيغة مختلفة.

تقادي الاستثناءات Catching Exceptions

يمكنك الكائن Err من تقادي 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:

```
Sub MySub()
    On Error GoTo ErrorHandler
    ...
    ...
    ' الشيفرات المتوقعة 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 المتوقعة حدوث استثناء بها
    ...
    ...
End Sub
```

استخدامك لأحد العبارات السابقة سينقل الاستثناء إلى الكائن Err والذي يحتوي على مجموعة من الطرق والخصائص المتعلقة بهذا الاستثناء، كخاصية 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
```

ملاحظة

لا يمكنك تفادي الاستثناءات بالصيغتين On Error ... و Try ... Catch ...
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

تدرك ورمي الاستثناء بالصيغ السابقة يعود اولا واخيرا على نوع الإجراء الذي امتلأ بالشفيرات المصدرية، ولا تستطيع 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 .NET Visual Basic، وذلك يعني ان لغات برمجة .NET الاخرى لا تتبع هذا الاسلوب وقد يؤثر على سير البرامج. قد لا يعينك الامر حاليا وذلك لانك لا تتوي تعلم لغة برمجة اخرى، ولكن لا تنسى ان فئاتك قد تستخدم من قبل مبرمجين اخرين يستخدمون لغات اخرى.

الفرق الجوهرى 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() واردة تداركه وطباعة مخرجاته على الشاشة ان وقع، ورغم ان الاستثناء وقع فعلا في الإجراء الآخر BBB() الا ان عملية قفص 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 المستدعي 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
```

يفضل رمي الاستثناء في حاله 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 تقيّدك في عمليات تنقيح برامجك بـ Visual Basic .NET.

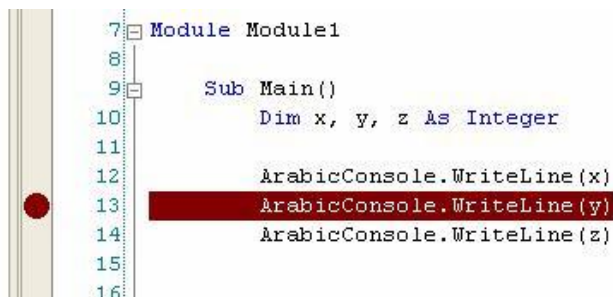
أساليب التنفيذ

بشكل تقليدي، يمكنك تنفيذ البرنامج كما فعلنا منذ الصفحة الأولى 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 السابقة او اضغط على المفاتيح [F5] + [Ctrl]، مع العلم ان خدمات التنقيح لن تكون مدعومة لحظة التنفيذ.

ملاحظة

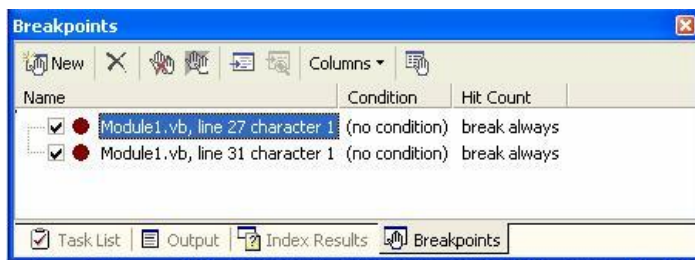
تنفيذ البرنامج باختيار الامر Start without Debugging سيفقد بيئة التطوير السيطرة على البرنامج ولن تتمكن من إيقافه من داخل بيئة التطوير (باختيار الامر Stop Debugging من القائمة Debug)، أي - بعبارة أخرى- سيتم تنفيذ البرنامج بشكل مستقل كما تنفذه من مستكشف النظام Windows Explorer.

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 6-7).



شكل 6-7: 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
```

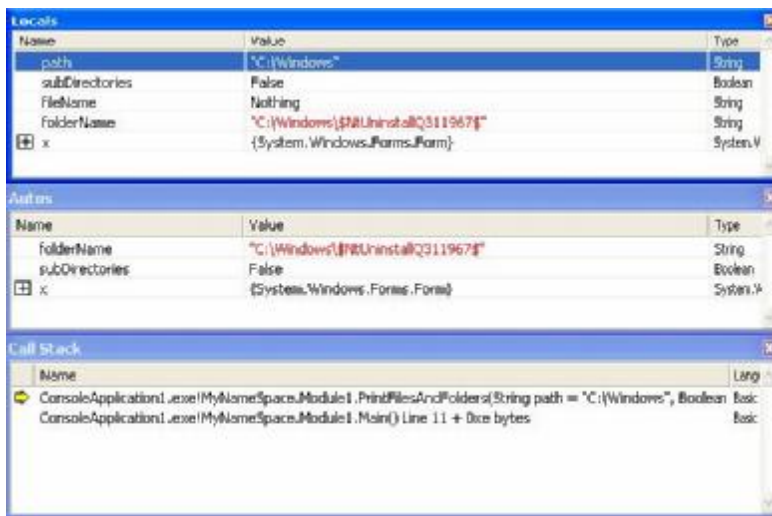
شكل 7-8: التنفيذ سطرا سطرا Step Into.

ملاحظة

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

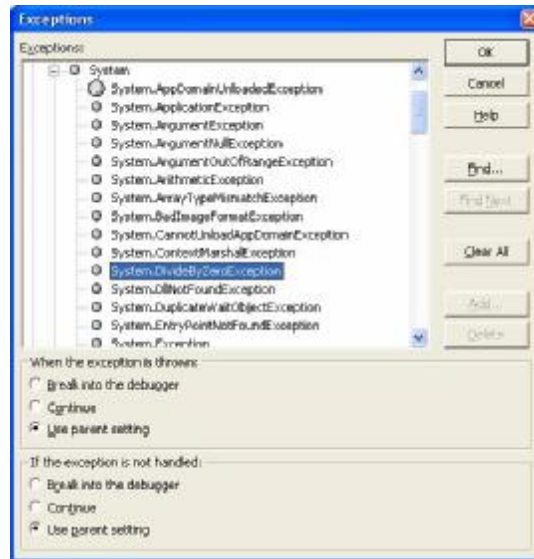
نوافذ أخرى

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).



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

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

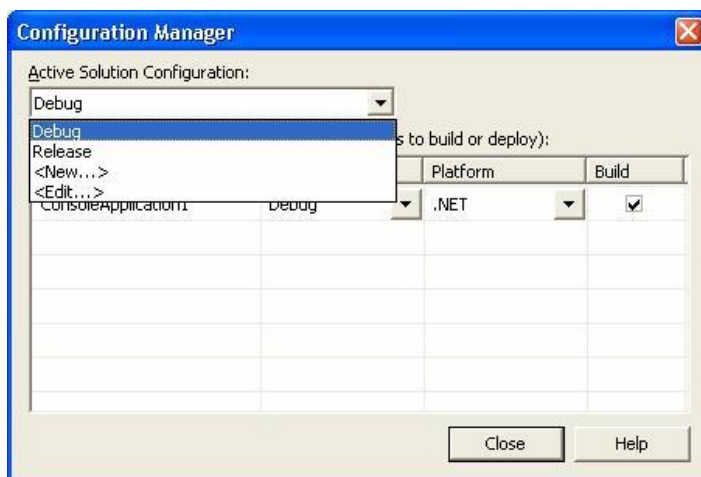


شكل 7-10: صندوق حوار الاستثناءات Exceptions.

الاعدادات Configurations

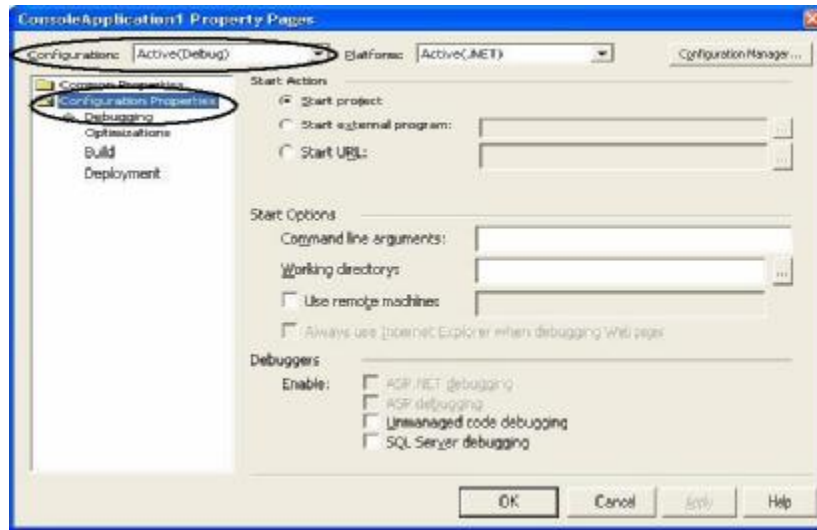
في الفقرة الاخيرة من الفصل الاول لمحت إلى الاعدادات Configurations وذكرت انها مجموعة من الخيارات الخاصة بعملية الترجمة Compiling للبرنامج. في كل مشروع جديد تنشئه، تقوم بيئة التطوير برفاق اسمين من الاعدادات هما Debug و Release، تستطيع الاختيار بينهما من القائمة الموجودة في شريط الادوات الرئيسي (شكل 1-13 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).



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

بعد إنشاء اعداد جديد او تعديل الحالي، يمكنك تغيير مواصفات الاعداد وتكوينات الترجمة عن طريق صندوق حوار خصائص المشروع Project Property Pages والانتقال إلى خانة التنبويب 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 إطار عمل .NET Framework على مجموعة من الفئات التي تمكنك من ادارة وتحرير ملفات ومجلدات الجهاز، بدءاً من العمليات البسيطة كالنسخ، النقل، او الحذف، وانتهاء بتحرير محتويات الملفات برمجياً. قسمت هذا الفصل القصير الى اربعة اقسام: القسم الأول يتعلق بالمجلدات واستخدام الفئة Directory، والثاني بالملفات والتعامل مع الفئة File، والثالث يشرح طريقة تحرير الملفات ومقدمة الى وحدات التخزين Streams، اما القسم الاخير فيتطرق الى ثلاث فئات هي Path، DirectoryInfo، و FileInfo بشكل سريع.

ملاحظة

لاختصار الشيفرات البرمجية، سأفترض في هذا الفصل انك قمت باستيراد مجال الأسماء التالي:

```
Imports System.IO
```

الفئة Directory

تحتوي الفئة Directory على مجموعة من الطرق المشتركة Shared Methods يمكنك 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")
```

ملاحظة

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

```
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")
```

يمكنك استخدام الطريقة Move() أيضا 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"))
```

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

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

أما الطرق `GetFiles()`، `GetDirectories()`، `GetLogicalDrives()` و `GetFileSystemEntries()` فهي تعود بمصفوفة من النوع `String` تمثل -على التوالي- أسماء محركات الأقراص الموجودة في الجهاز، المجلدات الموجودة في المسار المرسل، الملفات 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
```

أخيراً، الطريقة `GetParent()` تعود بالدليل الأبوي للمسار المرسل، نوع القيمة التي تعود بها هي كائن من الفئة `DirectoryInfo`، سأطرق لهذه الفئة في القسم الأخير فئات أخرى من هذا الفصل.

البحث عن الملفات والمجلدات

باستخدام الطرق السابق ذكرها، يمكنك الاستعلام والبحث عن كافة أسماء وخصائص الملفات الموجودة في القرص. طورت الإجراء `PrintFilesAndFolders()` التالي ليطلع كافة أسماء الملفات والمجلدات الموجودة في المسار المرسل، استخدمت أسلوب الاستدعاءات التراجعية **Recursion** في هذا الإجراء ليشمل السرد الملفات الموجودة في المجلدات الفرعية أيضاً. يمكنك تطوير هذا الإجراء أيضاً 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)

```

اما ان اردت الحصول على مواصفات الملفات، فيمكنك استدعاء الطريقة
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
...
...

```

وعلى العكس، فإن 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)

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

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

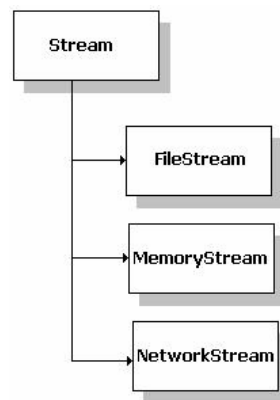
ملاحظة

الطريقتان GetAttributes() و SetAttributes() تعملان مع المجلدات Folders ايضاً، 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.



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

انظر ايضا

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

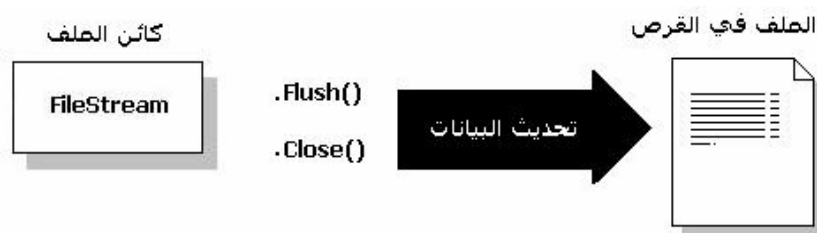
الخصائص والطرق المشتركة

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.

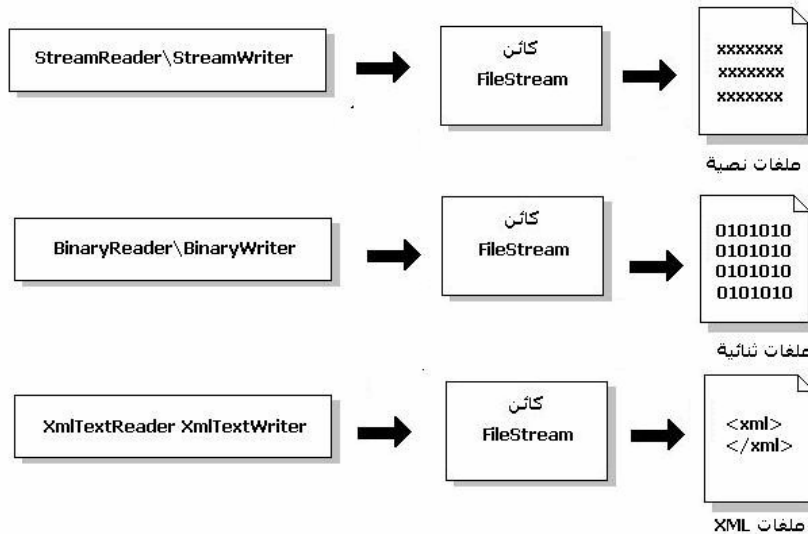
اما الحديث عن الطرق، فتوجد مجموعة من الطرق كالطريقة Read() التي تعود بمصفوفة من النوع Byte Array تمثل البايتات التي تم قراءتها، والطريقة ReadByte() التي تقرأ بايت واحد فقط. وعلى العكس، الطريقة Write() التي تمكنك من كتابة مجموعة من البايتات في مصفوفة من النوع Byte Array الى وحدة التخزين، والطريقة WriteByte()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

ملاحظة

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

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 Double، Integer... 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 بإنشاء اي كائن سهمما 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 StreamWriter، StreamReader، BinaryReader، و BinaryWriter. وقد نستخدم الفئتين XmlTextWriter و XmlTextReader 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 وتتشي كائن منها لإنشاء ملفات نصية والكتابة فيها، لديك العديد من الطرق لإنشاء 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 الكتابة 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)
```

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 الأرجح- 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 المخفوظة")
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 يمكنك من فتح ملف للقراءة، وهذه أسهلها:

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

بعد فتح الملف، فسيكون لديك عدة طرق يمكنك من قراءة البيانات بـصور 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.ReadToEnd)
...
textFile.Close()
```

ملاحظة

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

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:

```
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 المšíد، 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)

' قيمة من النوع Integer
binaryFile.Write(10)
' قيمة من النوع Double
binaryFile.Write(200.5)
' قيمة من النوع Boolean
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)

' قيمة من النوع Integer
ArabicConsole.WriteLine(binaryFile.ReadInt32())
' قيمة من النوع Double
ArabicConsole.WriteLine(binaryFile.ReadDouble())
' قيمة من النوع Boolean
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 المرسله، 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 زعر")
```

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 زعر
```

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.

لذلك 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.

ملاحظة

كإضافة لثقافتك البرمجية، يسمى الأسلوب الأول **بالحروف مسبقة الحجم** Prefix-length Strings أما الأسلوب الثاني فيعرف **بالحروف ثابتة الحجم** Fixed-length Strings. أتمنى منك التمييز بين هذه المصطلحات إن رأيتها يوماً في مستندات .NET Documentation.

تكوين Custom Streams خاصة

الفئات السابقة التي استخدمناها (StreamReader، StreamWriter، BinaryReader... الخ) تعمل بشكل رائع مع أنواع البيانات القياسية (Integer، String، Boolean، Double وغيرها) ولكن إن حاولت استخدام أنواع 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 العميري"
Turki.Age = 99

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

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

في مثل هذه الحالات، قد تطور فئات خاصة لتمكنها من التعامل مع أنواع بيانات أخرى 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 = "تركبي 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
```

```
Turki = MyFile.ReadPerson

ArabicConsole.WriteLine(Turki.Name)   ' تركي العمري
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 واطافة المزيد من الشيفرات المصدرة والخاصة بالكتابة او القراءة.

فئات أخرى

قبل ان اختتم 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 مثلاً هذه الخصائص التي تعود بالحروف المستخدمة في تحديد فواصل المسارات (قد تختلف النتائج عند تطبيق الشيفرة التالية على أنظمة تشغيل أخرى غير Windows):

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

المزيد ايضاً، هذه مجموعة اضافية من الطرق تمكنك من استخلاص جزءاً معيناً من 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 FileInfo و DirectoryInfo

تمثل الفئة DirectoryInfo مجلد معين، والفئة FileInfo ملف معين، لإنشاء كائنات من هذه 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 وراثياً 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 الأخرى. هذا مثال يشرح استخدام الطريقة GetDirectories() التي تعود بكائنات (من النوع DirectoryInfo) تمثل المجلدات الفرعية التابعة للمجلد الذي يمثلته الكائن:

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

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

أما الفئة FileInfo فهي أيضاً تحتوي على مجموعة من الطرق والخصائص الخاصة بطبيعة عملها كالخاصية Length التي تعود بحجم الملف، ومجموعة من الطرق الإضافية التي تستخدم لفتح وإنشاء الملفات. هذا مثال آخر 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 subfile و subfolder.

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.

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 Instance Object 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 تجدوها في مجالات الاسماء التالية:

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

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

```
Imports System.IO
```

مدخلك إلى تسلسل الكائنات

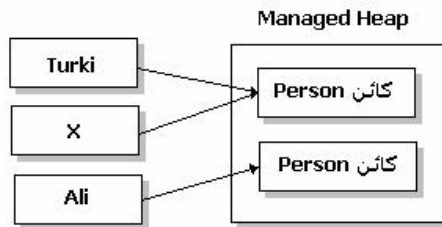
سأبدأ معك هذا الفصل بهذا القسم الذي يعتبر مدخلا الى التسلسل، حيث سنبدأ بتعريف عملية التسلسل في عالم .NET. ثم نقوم بتطبيقه بشكل عملي وذلك بعرض مثال بسيط يستخدم مفهوم التسلسل لحفظ الكائن في ملف على القرص الصلب واستعادته مرة أخرى، يلي ذلك عرض لكيفية تطبيق التسلسل على الفئات التي تعرفها بنفسك، وذلك باستخدام 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 انشاؤه من فئة، وهي لا تمثل متغيرات الكائن، اذ ان نسخة الكائن قد يشير لها أكثر 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 لاحقاً إعادة احياء الكائن ورافقه بمؤشر في شيفرتك وذلك **بعكس عملية التسلسل Deserializing**.

من الآن وصاعداً -يفضل التسلسل- لن نخشى على كائناتك من الموت أو الضياع، إذ إن بقاء الكائن على قيد الحياة لن يعتمد بعد الآن على خروج مؤشر الكائن من مده Scope سواء كان ذلك بنهاية الإجراء، أو بالإنتهاء الكلي للبرنامج.

يلعب التسلسل دوراً حيوياً في تطبيقات إطار عمل .NET Framework، ولا يقتصر استخدامه على حفظ الكائنات في وحدات تخزين كملفات في الأقراص، بل أنه يتجاوز ذلك بمرحلة، إذ أن هذه التقنية تمكن المبرمجين من تبادل كائناتهم بين البرامج المختلفة، أو حتى بين دول العالم المتباعدة عن طريق شبكة الانترنت باستخدام خدمات Web Services أو تطبيقات ASP.NET. وقد اشررت في الفصل السابع **اكتشاف الأخطاء** إلى أن الاستثناءات الخاصة Custom Exceptions يجب أن تكون قابلة للتسلسل Serializable لتتمكن من رميها بين المجموعات المختلفة.

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

التسلسل بالصيغة الثنائية Binary Serialization

في البداية دعني أطمئنك إلى أن إطار عمل .NET Framework يعرف جيداً كيف يجري التسلسل على أنواع البيانات الأولية (Primitive Types كـ String، Integer، Double، Boolean... الخ)، فكل ما تحتاجه لتطبيق التسلسل على هذه الأنواع 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 أن 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 الآن 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() = {"عباس", "برعي", "حسنه"}
Dim st As FileStream = File.Open("C:\test.dat", FileMode.OpenOrCreate)
Dim SerialObj As New BinaryFormatter()
```

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

```
' اغلق وحدة التخزين Stream
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):



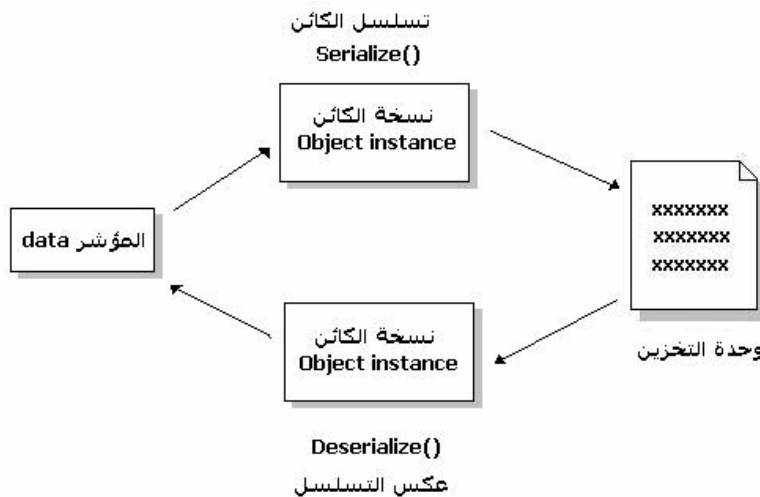
```
Dim data As String()
Dim st As FileStream = File.Open("C:\test.dat", FileMode.OpenOrCreate)
Dim SerialObj As New BinaryFormatter()
```

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

```
ArabicConsole.WriteLine(data(0)) ' حسونه
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

يعكس العملية ونقلنا نسخة الكائن من وحدة التخزين الى المؤشر data باستدعاء الطريقة `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 نعرفها بنفسك باستخدام التركيب `Class ... End Class` كما في الفقرة التالية.

تسلسل أنواع بيانات مخصصة (غير قياسية)

تطبيقاً، كل ما تحتاجه لجعل فئاتك قابلة للتسلسل هو استخدام الموصوفة `Serializable` Attribute، وجعلها في اعلى شيفرة تعريف الفئة فقط:

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

ما يجب عليك معرفته هو ان جميع الحقول التابعة للفئة (حتى وان كانت خاصة Private) والمتغيرات الاستاتيكية Static Variables داخل إجراءات الفئة سيطبق عليها التسلسل، ولكي تتحقق من ذلك اصف هذه الشيفرة بين فكي التركيب Class ... End Class:

```

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

    Private m_Age As Integer   ' متغير خاص Private
    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    ' متغير ستاتيكي Static

        x += 1
        Return x
    End Function
End Class

```

والآن ابدأ باستخدام الفئة السابقة، واسند أي قيم لحقولها وخصائصها ولا تنسى استدعاء 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 = "تركى العسيري"
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 العمري
ArabicConsole.WriteLine(Turki.Age)            ' 99
ArabicConsole.WriteLine(Turki.GetValue)       ' 4

st.Close()
```

ملاحظة

عند عكس عملية التسلسل لفئاتك الخاصة، فلا بد 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 يودي توضيحها عند استخدام الموصوفة Serializable Attributes: هذه الموصوفة 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 الموصوفة 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
```

خريطة الكائنات Object Graph

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

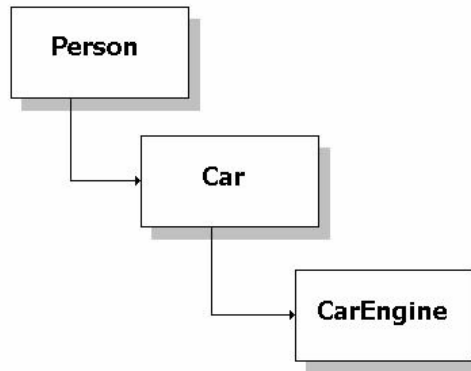
خريطة الكائنات Object Graph هي العلاقات Relationships والمراجع References الموجودة بين الفئات في الهرم الكائني Object Hierarchy، فلو كانت لدينا هذه الفئات الثلاث:

```
<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 الوراثة Inheritance.



شكل 9-3: علاقة الاختواء بين الفئات.

ملاحظة

خريطة الكائنات Object Graph تشمل الكائنات من النوع المرجعي Reference Type فقط، أما 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)
ArabicConsole.WriteLine(Test.Car.Name)
ArabicConsole.WriteLine(Test.Car.Engine.Cylinder)
...
...
...

st.Close()

```

تركي العسيري
BMW
12 v

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 أخرى إلى الكائن الأول Abbas، ومن ثم إلى Burey وهكذا إلى ما لا نهاية.

السيناريو السابق صحيح نظرياً، ولكن يسرني اخبارك بأن 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 تسلسلها مرة واحدة فقط، والشفرة 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

' لا توجد 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 بإمكانك 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
```

```

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

```

حسناً، 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 = "تركى العسري"
Turki.Car.Name = "BMW"
Turki.Car.Model = "Class 7"

Turki2 = Turki.Clone

ArabicConsole.WriteLine(Turki2.Name)           ' تركى العسري
ArabicConsole.WriteLine(Turki2.Car.Name)        ' BMW

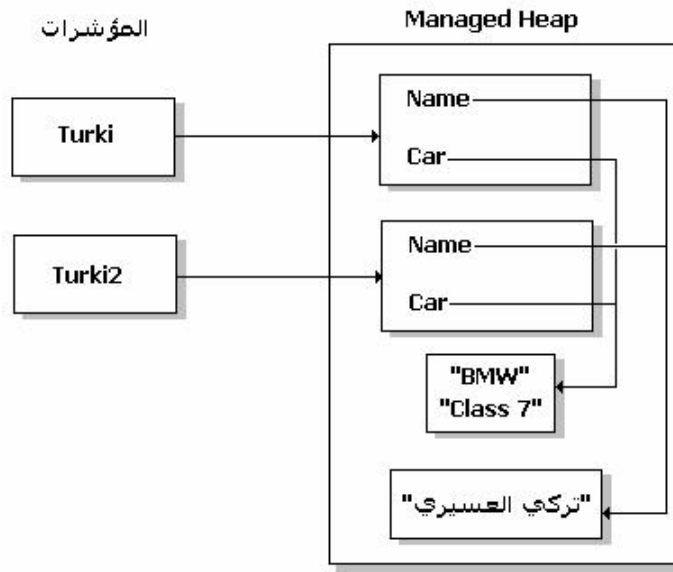
```

من المخرجات السابقة، يتضح لنا ان الكائن المحضون Car قد تم نسخه بالفعل، وبذلك يكون مؤلف الكتاب مخطئاً 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 ان الكائن لم يتم نسخه (شكل 9-4):

```

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

```



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، وإنما مؤشرات فقط. أما إن أردت نسخ جميع الكائنات الموجودة في خريطة الكائنات، فإنك تريد تطبيق ما يسمى **بالنسخ العميق Deep Copying**. لعمل ذلك، لن تجد أسهل من استخدام التسلسل، أعد كتابة الطريقة Clone() في الفئة Person بهذا الشكل (ولا تتسبى استخدام الموصفة Serializable Attributes في الفئات Person و Car):

```
<Serializable(> _
Class Car
...
...
End Class

<Serializable(> _
Class Person
    Implements ICloneable
...
...
```

```

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 المحفوظة في الكائن:



```

Dim Turki As New Person()
Dim Turki2 As Person

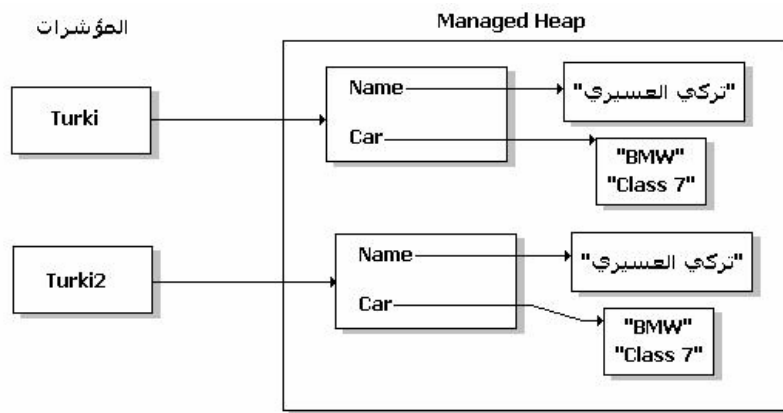
Turki.Name = "تركبي العسكري"
Turki.Car.Name = "BMW"
Turki.Car.Model = "Class 7"

Turki2 = Turki.Clone

ArabicConsole.WriteLine(Turki2.Name)           ' تركبي العسكري
ArabicConsole.WriteLine(Turki2.Car.Name)       ' BMW

ArabicConsole.WriteLine(Turki.Car Is Turki2.Car) ' False ٢ اثم 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 حق، 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 ستري 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 تلقائياً بمجرد تطبيق التسلسل على الفئة، استخدم الوسيطة المرسلة الاولى `info` ان اردت اضافة بيانات تود تضمينها في تسلسل الكائن الحالي باستخدام طريقته `: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` ايضاً 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 واطافة مشيد قابل للوصول حتى تتمكن من انشاء كائن من الفئة:

```
Public Sub New()
    ...
End Sub
```

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 الاجراءGetObjectData() حيث يتم استدعائه لحظة التسلسل، والمشيّد 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) Implements
        ISerializable.GetObjectData

        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 مقبلاً 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 إطار 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 سنكشف 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، فليست مضطراً 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) واستخدمها 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 = "تركى العسيري"
Turki.Age = 99

SerialObj.Serialize(st, Turki)

st.Close()
```

الشفيرة السابقة ستقوم بإنشاء 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>تركى 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.

كما فعلنا سابقاً مع الفئة `BinaryFormatter`، يمكنك عكس عملية التسلسل -ان كان بصيغة XML- باستدعاء الطريقة `Deserialize()` التابعة 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 إضافية

يوفر لك إطار 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`، بالإضافة 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>
```

الموصوفة XmlElement Attributes:

عند تطبيق التسلسل على كائن بصيغة XML، فسيتم حفظ اسماء حقوله في وسوم XML Tags كما هي موجودة بالشفيرة المصدرية. اما ان رغبت في تغيير اسم الحقل عند تسلسله بالصيغة XML، فاستخدم الموصوفة 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>
```

الموصوفة XmlAttributeAttribute Attributes:

تمكنك الموصوفة 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 جنباً الى جنب عند التعامل مع الحقول من نوع المصفوفات (وبالتحديد مصفوفات الكائنات)، حيث تستخدم المواصفة الاولى XmlArray Attributes لتعريف اسم عنصر XML Element 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 بإنشاء كائن من الفئة Customer السابقة، واسند قيم لجميع خصائصها وحاول تطبيق التسلسل على الكائن بصيغة XML في ملف خارجي، ليظهر لك شيئاً مثل (لاحظ الوسوم بالخط السميك Bold لتعرف كيف تؤثر المواصفتان XmlArray و 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:

اخيراً، 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
```

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 بصيغة 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 اطلاقه 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 شيئاً 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 يدوياً ان كانت العناصر غير معرفة. مثلاً، 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 شيئاً 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>
```

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

عند عكس التسلسل على الملف السابق، سيتم استعادة الحقل Age بشكل صحيح، اما الحقل Name فلن يحتوي على شيء والسبب واضح. لذلك، عليك قراءة العنصرين في ملف XML ووضعهما في المكان المناسب، اكتب اجراء جديد باسم UnknowXMLElemntEvent مثلاً 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 فنص 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 العمري
ArabicConsole.WriteLine(Test.Age) ' 99

st.Close()
```

هذا ليس كل شيء، ولا أريد منك أن تعتقد أنك عرفت الكثير عن تسلسل الكائنات Object Serialization بعد قراءتك للصفحات السابقة، حيث إن هذا الفصل لا يعتبر الا مقدمة بسيطة الى التسلسل وطرق تطبيقه بـ Visual Basic .NET. إن 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

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 مثلاً).

لن أنتقل إلى التفاصيل الأخرى حتى أتأكد من فهمك واستيعابك لمسارات التنفيذ، وقد يقرب لك هذا المثال مفهوم مسارات التنفيذ بصورة أفضل، 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، فلا بد ان تتم كل عملية بعد نهاية الأخرى -أي لا تستطيع إجراؤها 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 المصدريّة 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 تنفيذيين 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 تنفيذيين 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 الخاصة 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 ستري 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

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. وسأخذ هاتين الحالتين التان تبيان 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 $(10 + 20) \div 2 = 15$ 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 $(1 + 11) \div 2 = 6$ 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، سنكتشف 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.

حكمة برمجية وضعها دائما في عين اعتبارك: كلما زاد عدد مسارات التنفيذ في شيفراتك، كلما زاد تعقيدها وصعوبة اكتشاف أخطائها.

في إطار عمل .NET Framework، مسار التنفيذ ما هو إلا كائن منشأ من الفئة Thread يتطلب مشيده إجراء مفوض Delegate باستخدام الكلمة المحجوزة AddressOf (لأبدن ان 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.WriteLine($"c")
    Next

End Sub

Sub DoSomething()
    Dim counter As Integer

    For counter = 1 To 1000
        Console.WriteLine($"c")
    Next

End Sub

```

مخرجات الشيفرة السابقة، ستكون شيئاً شبيهاً بـ:

```
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$ *****  
***** $$$$$$$$$$$$$$$$$$  
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$ ***** $$$$$$$$$$$$ ...
```

تعتمد في المثال السابق استخدام الكائن Console عوضا عن ArabicConsole، وذلك لمشاكل ستفهمها لاحقا في هذا الفصل.

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 أن أذكرك هنا بأن السطور التي تلي الاستدعاء `newThread.Start()` سيتم تنفيذها أيضا في نفس الوقت.

الطرق والخصائص

إن كانت الطريقة `Start()` تشغل مسار التنفيذ، فإن 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() ' إيقاف مسار التنفيذ الحالي
        End If
        Console.WriteLine("*" & c)
    Next
End Sub
```

الطريقة `Abort()` توقف عمل مسار التنفيذ وتنتفيه من الحياة بشكل نهائي، ولكن توجد طريقة أخرى توقف عمل مسار التنفيذ بشكل مؤقت ولا تنتهيه وهي `Suspend()`، والتي قد تحتاج إتباعها بالطريقة `Resume()` لتكمل عمل مسار التنفيذ:

```
Dim newThread As New Thread(AddressOf DoSomething)
...
...

' تشغيل مسار التنفيذ
newThread.Start()
...
...

' إيقافه 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:

ايقاف مسار التنفيذ الحالي لمدة نصف ثانية ' Thread.CurrentThread.Sleep(500)

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، فستطيع 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

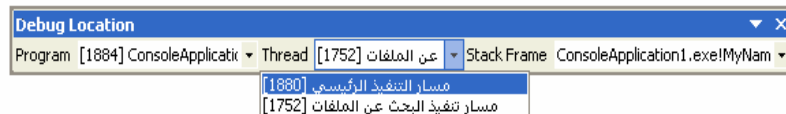
من الغباء الاستعلام عن الخاصية IsAlive لمسار التنفيذ الحالي:

```
If Thread.CurrentThread.IsAlive Then ...
```

فهي ستعود بالقيمة True دائما لأسباب منطقية.

الخاصية Name حرفية من النوع String قابلة للقراءة والكتابة، وفي الحقيقة ليس لها أي تأثير عملي على مسار التنفيذ لحظة التنفيذ، ولكنها تفيدك كثيرا لحظة التتبع 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، يمكنك 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:

الحالة	القيمة
انهي مسار التنفيذ بشكل نهائي.	ThreadState.Aborted
تم طلب إيقاف لمسار التنفيذ.	ThreadState.AbortRequested
مسار تنفيذ خلفي BackGround	ThreadState.BackGround
.Thread	
يعمل.	ThreadState.Running
الإيقاف المؤقت.	ThreadState.Suspended
تم طلب إيقاف مؤقت لمسار تنفيذ.	ThreadState.SuspendedRequested
لم يبدأ بعد.	ThreadState.UnStarted
تم استدعاء الطريقة Join على مسار التنفيذ.	ThreadState.WaitSleepJoin

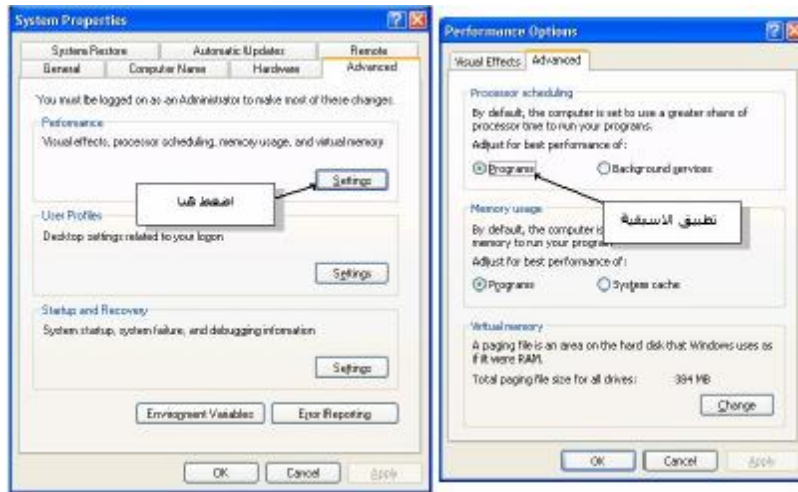
خاصية الأسبقية Priority:

أنت تعلم وأنا أعلم ان نظام التشغيل Windows هو نظام تشغيل متعدد المهام **Multitasking**، بحيث يمكنك من تشغيل أكثر من برنامج في وقت واحد، الأمر الذي يفرض على النظام توزيع المعالجة بين التطبيقات المختلفة. في الإصدارات السابقة من نظام التشغيل Windows (الإصدار 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 شوائله 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 اعتمده مطورو نظام التشغيل Windows هو إنتاج 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، الأمر الذي يترتب عليه نتائج إيجابية 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 Settings -> Performance -> Advanced -> System -> Control Panel (شكل 10-2).



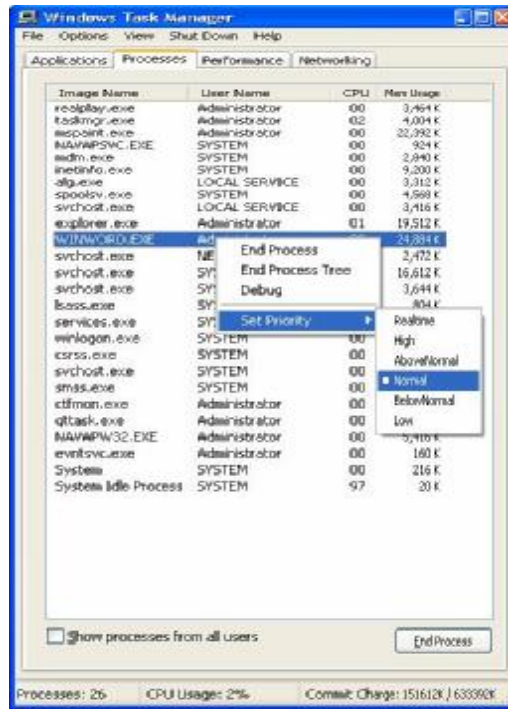
شكل 10-2: نظام التشغيل يعطي وقت معالجة أكثر للبرامج النشطة.

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.



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 عدد 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 عدد 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
```

عند تنفيذك للشفرة السابقة، ستفاجئ إن اكتشفت أن مسار التنفيذ الأول قد أنهى حلقة التكرارية (والتي عددها 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 إنهاؤها 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 تبقى 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
```

أما إن كان مسار التنفيذ الثاني من النوع الخلفي Background Thread، فسيتم إنجازه بمجرد 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
```

ملاحظة

لا تحاول استخدام الكائن ArabicConsole عند تجربة الشيفرة السابقة، وذلك لأن البرنامج لن ينتهي بسبب أن ArabicConsole يحتوي على مسار تنفيذ خاص به يبقى البرنامج على قيد الحياة.

التعامل مع مسارات التنفيذ

بادئ ذي بدء عليك معرفة أن مسارات التنفيذ تعمل كالبرامج المستقلة، فيمكنك اعتبار أن كل مسار تنفيذ هو برنامج عامل، لدرجة أنه لو حدث استثناء Exception في مسار تنفيذ، فلن تتأثر مسارات التنفيذ الأخرى وستكمل عملها كأن شيئاً 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 أن مسار التنفيذ الثاني قد رمى استثناء Throw Exception، إلا أن مسار التنفيذ الرئيسي سيكمل عمله بشكل طبيعي. مع ذلك، أنصحك بشدة بتقادي 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 شيءًا 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 يمكنك 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")) ' 100.00 رس
    End Sub

    Sub DoSomething()
        MsgBox(100.ToString("C")) ' 100.000 ج م
    End Sub
End Module
```

انظر أيضا

موضوع الاعدادات الإقليمية حديث ذو شجون لمحت إليه سابقا في
الفصل السادس الغثات الأساسية.

على صعيد آخر، عمليات الإيقاف النهائي والإيقاف المؤقت باستخدام الطرق () Abort و Suspend() لا تتم فورا لحظة استدعائها، وإنما 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 أمنيته عرض مثلا يوقف مسار تنفيذ وهو ليس في لحظة نقطة آمنة. ولكني -والاعتراف بالحق فضيلة - لست متأكدا حتى هذه لحظة من مدى استيعابي للنقاط الآمنة. لذلك، دعني أكرر كلمتي الأخيرة وأنصحك بالانتقال إلى مكتبة MSDN لعل وعسى تطلع منها بشيء.

أما 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- إنها تموت إن ماتت جميع مسارات التنفيذ العادية في البرنامج بشكل تلقائي.

حالة غريبة حدثت لي عندما لعبت قليلا مع مسارات التنفيذ لم أجد تفسيراً 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
```

سنتهي 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، سنضطر إلى استخدام التزامن Synchronization حتى تتقي شر التعارضات التي قد تحدثها لك مسارات التنفيذ المتعددة.

المتغيرات المحلية الديناميكية

جميع المتغيرات بكافة أنواعها (من منطلق قابلية رؤيتها Visibility وعمرها Lifetime) مشتركة بين مسارات التنفيذ، واعني بكلمة مشتركة -في هذا السياق- انها قابلة للوصول من جميع مسارات التنفيذ التي تعرفها في البرنامج. مع ذلك، المتغيرات المحلية الديناميكية 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 علي ديناميكي  
    Dim X As Integer  
    ' متغير علي 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 المسارين 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 علي ديناميكي
    Dim X As Integer
    ' متغير علي سنايكي
    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()
    End Sub
End Module

```

```

        Console.WriteLine(X) ' 200
        Console.WriteLine(Y) ' 100


    End Sub

    Sub DoSomething()
        X = 200
        Y = 200
    End Sub
End Module

```

قد يسرك كثيرا استخدام الموصفة ThreadStatic Attribute وتجنب عشرات المشاكل والأخطاء التي تحدث بسبب التعارض على المتغيرات من قبل مسارات التنفيذ المتعددة، ولكن يؤسفني إخبارك ان الموصفة 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 للشفيرة المصدرية التالية، أصبت بخيبة أمل كبيرة بعد رؤيتي لقيمة المتغير X:

```


Class TestClass
    ' لن يفيدك استخدام الموصفة هنا
    <ThreadStatic()> Public X As Integer
    ' ستعمل بكفاءة على هذا الحقل لانه مفتوك
    <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

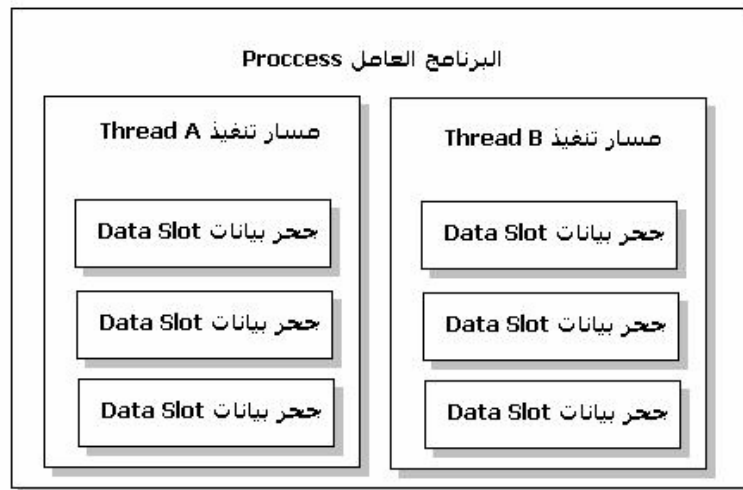
    End Sub

    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 **جذر 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**، و**جحور 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 جحور بيانات جديد، استدعي الطريقة **AllocateNamedDataSlot()**:

إنشاء جحور بيانات مسمى '
`Thread.AllocateNamedDataSlot("Name")`

الطريقة السابقة تعود بكائن من النوع LocalDataStoreSlot، لذلك يفضل تعريف متغير من هذا النوع حتى يسهل عليك التعامل مع جحر البيانات لاحقاً:

```
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 العسوي")
```

وعلى العكس، يمكنك الطريقة GetData() من قراءة جحر بيانات مسمى:

```
Console.WriteLine(Thread.GetData(namedDataSlot)) ' تركي العسوي
```

وفي حال ما إن تم فقدان المؤشر namedDataSlot 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 العسوي
```

أخيراً، لا تنسى إلغاء جحر البيانات عند عدم الحاجة إليه 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.


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 جحور البيانات، بل سنكتفي باستخدام مؤشر من النوع LocalDataStoreSlot لتمييز جحور 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 DoSomething():



```
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 آخر، حيث 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 المصدريّة على بيانات مسارات التنفيذ الأخرى في البرنامج. ولكن توجد 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 فنصه 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 المسألة بعين الاعتبار عند إطلاق الأحداث.

التزامن Thread Synchronization

في هذا القسم أحاول عرض لك بضعة أساليب يمكنك 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 تنفيذيين، 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
```




متغير مرجعي عام ومشارك 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.WriteLine(Thread.CurrentThread.Name)
```

Next

```
Console.WriteLine()
```

End SyncLock

End Sub

بعد إضافة التركيب السابق، سيضمن لك Visual Basic .NET بأن 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 تنظيماً:

From Thread 1

111 ...

From Thread 5

555 ...

From Thread 3

333 ...

From Thread 2

[illegible]

...

...

Synchronization المواصفة

بدلاً 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 لا بد من اشتقاق الفئة

```
Inherits ContextBoundObject
```

...

...

End Class

يعيب الموصافة Synchronization هو ان الفئة التي تستخدمها لابد من تكون مشتقة وراثيا من الفئة ContextBoundObject مما يحرمك من إمكانية اشتقاق فئة أخرى. عيب آخر في الموصافة 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 الموصافة MethodImpl.

الموصافة MethodImpl

استخدامك للموصافة Synchronization يمنع جميع مسارات التنفيذ من تنفيذ شيفرة الكائن ان كان مشغول بأحد مسارات التنفيذ، وهذا بحد ذاته أسلوب غير مرن، حيث ما لفائدة من تعدد مسارات التنفيذ إن كانت العمليات لا تتم إلا من قبل مسار تنفيذ واحد فقط في وقت واحد. استخدامك للموصافة MethodImpl يمكنك من تحديد الأعضاء التي تود حكرها على مسار تنفيذ واحد، مع العلم انها تعمل بنجاح أيضا على الأعضاء المشتركة Shared Members. هذه طريقة استخدامها:

```
استد 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
```

يودي توضيح بعض المسائل المترتبة على هذه الموصافة، حيث ان عملية الإقفال لا تتم على الطريقة التي عرفت فيها الموصافة فقط، بل تشمل وتمتد إلى الطرق الأخرى، فمثلا لو استخدمت الموصافة 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 لأنني 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.

لست بحاجة لإنشاء كائن جديد من الفئة Monitor لاستخدامها، حيث أن جميع طرقها مشتركة Shared Methods، استدعي الطريقة Enter() لتحديد نقطة البداية للشفيرة المراد حصرها، والطريقة Exit() لتحديد نقطة النهاية، مع العلم أن 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 أن لم تكن قيمة الشرط صحيحة:

```
If Y = 0 Then Monitor.Exit (X)
```

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 كوسيلة 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) ستجرب 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 يمكنك 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:

```
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

الوسيلة 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:

```
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 لها 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 مسارات 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
```

أيهما 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، اصف 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: Windows.Forms.Timer، Threading.Timer، و System.Timers. في هذا القسم اعرض لك المؤقتين الأول والثاني، اما الاخير فأرى انه من المناسب تأجيله حتى الفصل الرابع عشر الأدوات Controls.

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 لمشيدته. الوسيطة الأولى تمثل الإجراء المفوض الذي يشترط أن يحتوي على وسيطة من النوع Object:

```
Dim MyTimer As New Threading.Timer(AddressOf DoSomething, ...)

Sub DoSomething(ByVal x As Object)
    ...
    ...
End Sub
```

يمكنك إرسال أي قيمة إلى هذا الإجراء المفوض، بذكرها كوسيطة ثانية لمشيد فئة المؤقت:

```
' ارسل القيمة 100
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 يمكنك من تعديل القيم السابقة، ولكن توجد الطريقة (`Change()`) التي يمكنك 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.

أؤكد لا اصدق أنني 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**.

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 لتركيبية 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 Visual Basic .NET (أو أي لغة .NET. أخرى)، 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 من وحدة مدارة)، هذه الوحدة المدارة يمكن ان تعمل بشكل مستقل أو تعتمد على وحدات مدارة أخرى 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 الملف خاص بنظام التشغيل Windows والذي تشمله جميع التطبيقات التي تعمل تحت بيئة Microsoft Windows.

(2) **.NET Framework File Header**: وهو أيضاً 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 المرسل بين المجمعات أو الوحدات المدارة المختلفة، كما يمكنك الاستعلام عن هذه البيانات باستخدام مجموعة من فئات الانعكاس Reflection Classes كما ستري لاحقاً في الفصل القادم.

(4) **شفيرة 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 شبيهة بلغة التجميع Assembly تسمى Microsoft Intermediate Language (MSIL أو IL). شيفرة MSIL تمثل شيفرة كل أجزاء الوحدة المدارة بعد الترجمة.

تطبيقاً، لن نستفيد الكثير من القسم الاول، الثاني، والرابع الا ان كنت تنوي التخصص في البنية التحتية لتطبيقات Windows، ولغة التنفيذ المشتركة CIL الخاصة بإطار عمل .NET. وهي مواضيع متقدمة جداً وخارج نطاق الكتاب، اما القسم الثالث Metadata فستري في الفصل القادم فئات الانعكاس **Reflection Classes** كيف يمكنك الاستفادة منها والاستعلام عن البيانات الموجودة في الوحدة المدارة بـ .NET Visual Basic، لتتمكن -مثلاً- من معرفة جميع الفئات والاستعلام عن جميع أعضائها من برنامج خارجي اخر.

المجمعات Assemblies

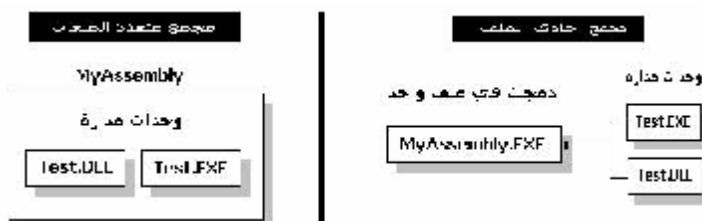
المجمع Assembly هو اصغر وحدة تمثل تطبيق منجز بأحد لغات .NET، أي بعبارة اخرى المجمع هو التطبيق الذي أنشأته، بحيث يكون له اسم خاص، رقم إصدار، اسم المصمم، الشركة... الخ. قد يكون المجمع في ملف قابل للتنفيذ مباشرة EXE، أو مكتبة DLL تستخدمها في برامج اخرى، أو مجموعة مترابطة من الملفات (EXE، DLL، DOC، HTML، 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، وإنما عليك إضافتها في خانة المراجع Reference حتى تتمكن من الاستفادة من فئاتها في برامجك الأخرى والتي تنجزها بلغات .NET الأخرى.

أما 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، ويمكن الفرق بينهما ان النوع الثاني يمكنك من استخدام الكائن Console والخاص بعرض المخرجات، أما الأول 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 والمشاركة

يوجد نوعان من المجمعات يمكنك إنجازها هما **المجمعات الخاصة Private Assemblies** و**المجمعات المشتركة Shared Assemblies**. المجمعات الخاصة هي مجمعات تضع ملفات 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 افترضنا هذه المجمعات:

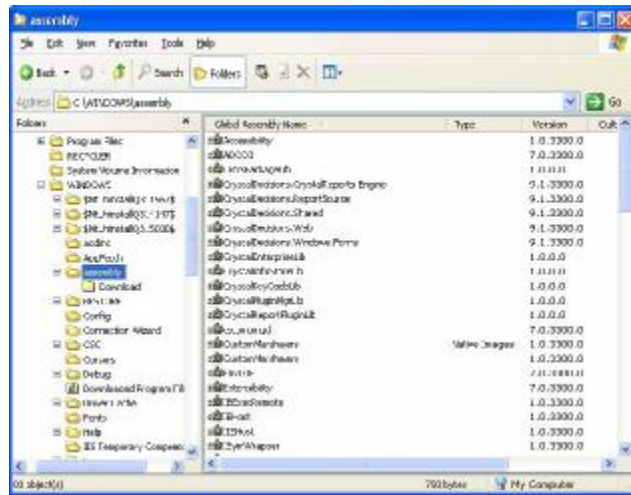
```
C:\Folder\AAA.DLL
C:\Folder\BBB.EXE
C:\Folder\CCC.EXE
C:\Folder\SubFolder\DDD.EXE
```

فيمكنك استخدام المجمع AAA.DLL من خلال التطبيقين BBB.EXE و CCC.EXE فقط، وذلك لأنهما في نفس مجلد المجمع الخاص أو في المجلد الأبوي له. مع ذلك، عليك استخدام الوسم <probing> في ملف التهيئة للتطبيق BBB.EXE لتحديد فيها اسم المجلد الفرعي والذي يحتوي على المجمع لإنجاز 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 - 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!**

أما المجمعات المشتركة فهي مجمعات قابلة للاستخدام من قبل مجمعات .NET. أخرى، وتضع ملفات 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 بأن مستكشف النظام Windows Explorer لا يريك الشكل الفيزيائي الحقيقي لملفات المجمعات المشتركة، فلو تستخدم موجه الأوامر Command DOS Prompt، ستكتشف ان المجمعات موزعة على شكل مجلدات، كل مجلد من هذه المجلدات يحتوي على مجلدات فرعية تمثل الإصدارات المختلفة للمجمع.

يمكنك حذف مجمع مشترك بالنقر بز الفأرة الأيمن على رمز المجمع واختيار الامر 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 الوحدة العامة للمجمعات GAC إلا إذا كانت لديك صلاحية المستخدم Administrator من قبل نظام التشغيل.

الأسماء القوية Strong Names

المجمعات الخاصة Private Assemblies يتم التفريق بينها على حسب موقع ملفاتھا في القرص، لذلك إن وجدت عشرات المجمعات المتشابهة في أسمائها وأرقام إصداراتها، فستتخصص المسؤولية عليك كمبرمج من وضع ملفات المجمعات في مجلدات منفصلة.

أما الحديث عن المجمعات المشتركة Shared Assemblies، فجميع المجمعات توضع فن نفس المجلد الرئيسي للوحدة GAC، وإن تشابهت أسماء المجمعات، فسيتم إنشاء مجلدات فرعية بأرقام الإصدارات المختلفة للمجمع، فلو تخيلنا أن المجمع AAA له إصدارين، 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 كافٍ 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 جداً (حجمه 128 بت) يتم إنشاؤه بخوارزميات معقدة بحيث تضمن عدم تعارض مفاتيحين عامين في الكرة الأرضية!

النقطة الهامة التي أريد أن أصل بك هي: اسم المجمع، مفتاحه العام، رقم إصداره، اسم الشركة المنتجة، أعدداته 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 لاحقاً.

المواصفة Assembly

يمكنك استخدام المواصفة Assembly إن أردت إسناد معلومات حول المجمع، كرقم الإصدار، اسم المجمع، الشركة، حقوق التأليف،... الخ:

```
Imports System.Reflection
Imports System.Runtime.InteropServices
```

```
<Assembly: AssemblyTitle("نظام الحاسبة")>
<Assembly: AssemblyDescription("برنامج حسابي لإدارة العمليات والحسابات")>
<Assembly: AssemblyCompany("شبكة المطورون العرب")>
<Assembly: AssemblyCopyright("© الحقوق محفوظة لشبكة المطورون العرب")>
<Assembly: AssemblyVersion("1.0.2321432")>
...
...
...
```

بالنسبة للإصدار، فيمكنك استخدام النجمة * حتى تزيد رقم المراجعة بشكل تلقائي في كل مرة تنفذ وتجرب المجمع:

```
<Assembly: AssemblyVersion("1.0.*")>
```

أخيراً، المشاريع التي تنجزها بيئة التطوير Visual Studio .NET تقوم بإضافة 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 إلا ملفات نصية منسقة بصيغ XML يمكنك تحريرها بأبسط برامج التحرير كالمفكرة Notepad لتغيير مجموعة من الإعدادات التي تتعلق بالمجمعات والربط بينها.

ملاحظة

إن 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.

أنواع ملفات التهيئة

في عالم NET. توجد ثلاث أنواع من ملفات التهيئة عبارة عن مستويات لتحديد الإعدادات 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، وإن أردت إنشاء ملف تهيئة التطبيق بنفسك، فلا بد أن 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

لا بد أن 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 أن قمت -مثلاً- بتوزيع أحد المجمعات المشتركة (نقل مكتبة فئات) على المستخدمين، وبعد فترة اكتشف خطأ في هذه المكتبة وأردت من المستخدمين إعادة تثبيتها، لذلك قد ترفض ملف تهيئة من هذا النوع حتى تعيد توجيه جميع التطبيقات التي استخدمت المجمع المشترك إلى الإصدار 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 إصدار 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 Documentation:

الوسم <requiredRuntime>:

عندما تنوي تنفيذ مجمعك على جهاز آخر، فإن إصدار إطار عمل .NET Framework الذي يتطلبه هو نفس الإصدار الذي تم ترجمته البرنامج فيه. مع ذلك، تستطيع تغيير الإصدار المطلوب باستخدام الوسوم <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 أن وسوم XML تفرق بين الحروف الصغيرة والكبيرة (أي أنها case-sensitive). لذلك، اكتبها بنفس حالة الأحرف التي تراها.

إن كان رقم الإصدار المحدد في الوسوم <requiredRuntime> يختلف عن رقم الإصدار الأصلي للبرنامج (أي الإصدار الذي تم ترجمة البرنامج عليه)، سيتم حفظ معلومات إضافية حول هذا الاختلاف في سجل النظام Windows Registry، وحتى لا تكثر هذه المعلومات في كل مجمع تقوم بتغييره، يمكنك إسناد القيمة True للخاصية safemode في الوسوم <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 إن كانت خاصة لعرض واجهات رسومية. مع ذلك، تتصح مستندات MSDN بإلغاء هذه الطريقة وإيقاف عمل كافة

مسارات التنفيذ لحظة تفريغ الذاكرة في ان كنت تتوي تطوير برامج تعمل في أجهزة الخوادم Servers ولا تحتوي على أية واجهات رسومية. يمكنك استخدام الوسم <gcConcurrent> في القسم الفرعي <runtime> لملف التهيئة وإسناد القيمة False لخاصيته enabled لوقف جميع 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 **الغثات والكائنات**.

الوسم <add>:

تستطيع استخدام الوسم <appSettings> لوضع قيم خاصة بك قد يحتاجها برنامجك (كمسار 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 المصدريّة لحظة تنفيذ البرنامج باستخدام الفئة System.Configuration.ConfigurationSettings.AppSettings بهذا الشكل:

```
Imports System.Configuration.ConfigurationSettings

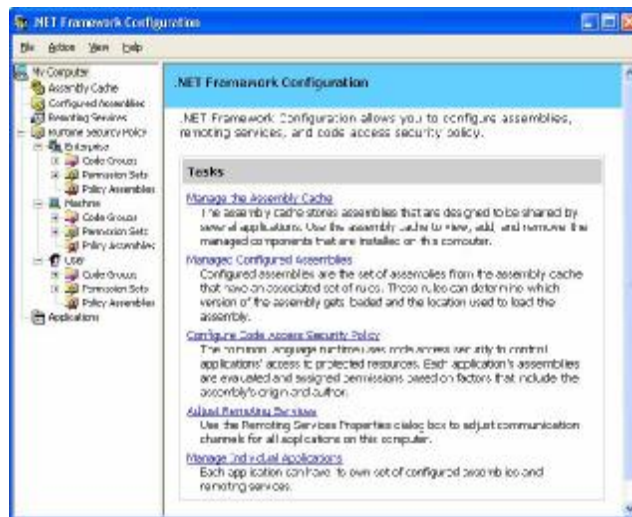
Module Module1
    Sub main()
        ' www.dev4arabs.com
        ArabicConsole.WriteLine(AppSettings("Site"))

        ' 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.

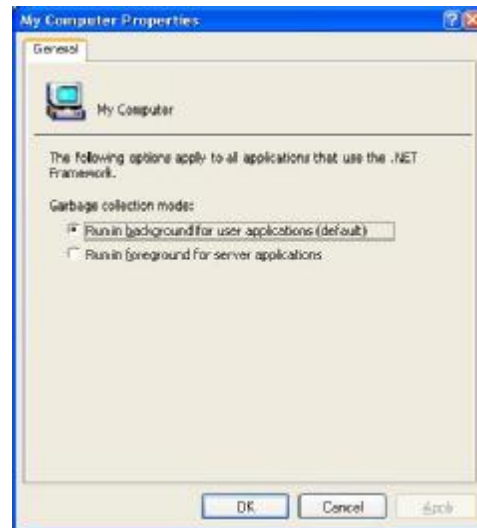


شكل 11-4: الأداة .NET Framework Configuration..

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، وWindows 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 4-11).



شكل 4-11: تغيير سلوك تنفيذ المجموعة Garbage Collection.

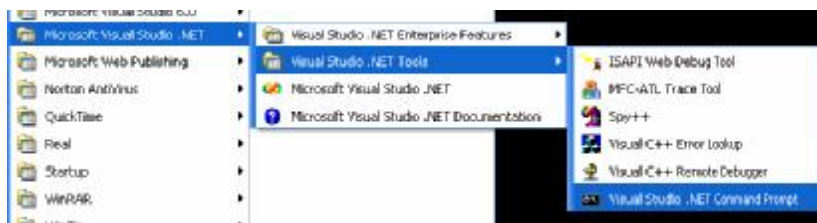
يمكنك إضافة وتصحيح ملفات تهيئة التطبيقات بالضغط على الرمز Application في الشجرة اليسرى للنافذة. وبالنسبة للمجمعات المشتركة، فستجد ضالتك بالضغط على الرمز Configured Assemblies.

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 عوضاً عن بيئة التطوير Visual Studio .NET، السبب في ذلك هو أن 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. أخرى، وغيرها من الأمور.

إن 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 إن كنت قد شغلت نافذة موجه الأوامر من خلال الرمز Visual Studio .NET Command Prompt الموجود في المجموعة البرمجية Visual Studio .NET Microsoft Visual Studio Start بقائمة (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.

المتراجم VBC.EXE

المتراجم 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:

```
' في الملف test.vb
Imports System

Module Module1
    Sub Main
        Console.WriteLine ("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، ولكنني استخدمت المصطلح وحدة مدارة لأبين لك أنه 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 الوسيلة /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 الوسيلة /target/ (يمكنك اختصارها إلى /t/):

```
(تطبيق winprog.exe)
C:\>vbc winprog.vb /target:winexe
```

```
(مكتبة mylib.exe)
C:\>vbc mylib.vb /t:library
```

استخدمنا المترجم VBC.EXE في الأمثلة السابقة لإنشاء وحدات ومدارة 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 الوسيلة /target/:

```
C:\>vbc test.vb /target:module
```

الامر السابق سينشئ ملف باسم test.netmodule يمثل شيفرات الوحدة المدارة فقط (ولا يشمل المجمع)، وان كانت الوحدة المدارة تعتمد أو تستخدم انواع بيانات لوحات مدارة اخرى، عليك اضافة هذه الوحدات المدارة باستخدام الوسيلة /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.

الرابط AL.EXE

الرابط AL.EXE هو رابط المجمعات Assembly Linker، وهو الوسيلة الوحيدة التي يمكنك من إنجاز 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.

المجمع myass

file1.netmodule

file2.netmodule

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 لأن 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 .NET Visual Basic ليست حساسة case insensitive.

مثال تطبيقي:

والان سأريك تطبيق عملي لإنشاء مجمع متعدد الملفات يشمل وحدات مدارة مترجمة من شيفرات مصدريّة بلغتين مختلفين هما .NET Visual C# و .NET Visual Basic. 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
```

الإجراء Sub Main() السابق يقوم باستدعاء الإجراء Sub2() والذي يعرض رسالة مفادها ان هذه شيفرة مكتوبة بلغة Visual Basic .NET، وبعد ذلك يقوم الإجراء باستدعاء الإجراء Sub3() وهو إجراء سنكتبه بلغة C#:

```
// في الملف file3.cs
namespace file3
{
    class clsfile3
    {
        static public void Sub3()
        {
            file4.clsfile4.Sub4();
        }
    }
}

// في الملف file4.cs
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 فوراً 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() (/out:myprog.exe)، واسم ملفه الرئيسي (/main:MainModule.Main):



```
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:

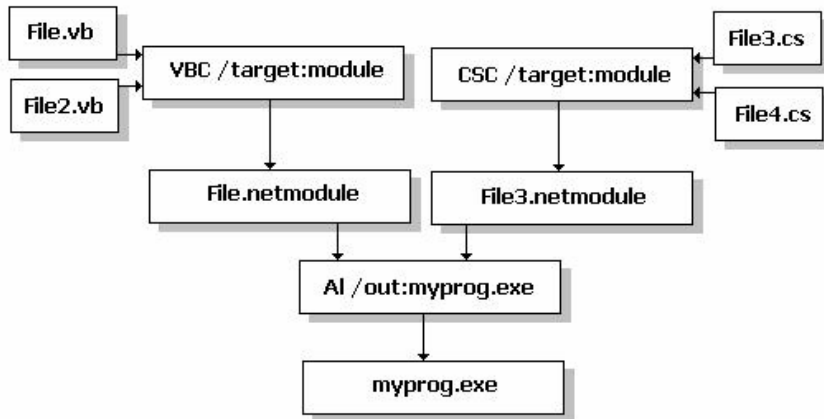
```
C:\test>myprog
```

```
VB code works.
```

```
C# code works.
```

```
C:\test> _
```

ان لم تفهم شيئاً 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 أسماء قوية Strong Names لمجمعاتك، الخطوة الأولى التي تحتاجها هي إنشاء مفتاح مركب Partial key وحفظه في ملف منفصل (باستخدام الوسيلة -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 لحظرة الترجمة أو الرابط:

```
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
```

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.

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 إنشاء 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
```

من الخصائص الأخرى، الخاصية EntryPoint والتي تعود بكائن من النوع Reflection.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 لأن 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 لاحقاً في هذا الفصل.

الطريقة GetType() 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.GetType()
    ArabicConsole.WriteLine(t.Name)
Next
```

الطريقة GetType() 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 و التابعة 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 الأنواع المعرفة في الوحدة المدارة وليس كامل المجمع.

التعامل مع أنواع البيانات

في هذا القسم اعرض لك المزيد من الطرق والخصائص التي يمكنك من الاستعلام عن الطرق والخصائص وكافة أعضاء البيانات التي تريدها لحظة التنفيذ. ولكن قبل ذلك، احتاج إلى تعريفك أكثر 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، وإنما 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 أكثر 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 `.NET Framework` الأخرى)

تحتوي على الطريقة GetType() والتي تعود بكائن من النوع Type يمثل نوع الكائن الحالي حتى لو كان الكائن منشئاً بالربط المتأخر Late Binding:

```
Dim t As Type
Dim obj As Object = New TestClass()

t = obj.GetType()

ArabicConsole.WriteLine(t.FullName) ' MyNamespace.TestClass
```

المزيد أيضاً، الطريقة GetType() السابقة تم إعادة تعريفها Overloads في الفئة System.Type بحيث يمكنك من إرسال قيمة حرفية من النوع String:

```
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-Sensitive، كما عليك استخدام الاسماء الأصلية لإطار عمل .NET Framework. (كـ Int32، Int64، DateTime.... الخ) عوضاً عن الأسماء الخاصة بلغة البرمجة .NET Visual Basic (كـ Integer، Long، Date... الخ).

بالإضافة إلى الطريقة GetType()، لديك الطريقة GetTypeInfo() والتي ترسل معها مصفوفة لكائنات، لتعود بنوع كل كائن من هذه المصفوفة على حدة:

```
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 اهنئك في اذنك وأخبرك ان الفئة Type يمكن لها ان تحمل كائناتها انواع بيانات لفئات منجزة بتقنية COM، تستطيع عمل ذلك باستدعاء الطريقة GetTypeFromProgID() وإرسال قيمة معرف ProgID (برمجة المكونات COM موضوع أقدم من ان يذكر في هذا الكتاب):

```
' 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 يمكنك من الحصول على تفاصيل أكثر للنوع المسند إلى الكائن من النوع Type، من هذه الخصائص الخاصة Name التي تعود بالاسم البرمجي للنوع، الخاصة FullName التي تشمل مجال الاسماء في الاسم البرمجي للنوع، الخاصة 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: IsInterface، IsClass، IsValueType، IsEnum 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 استخدم 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.

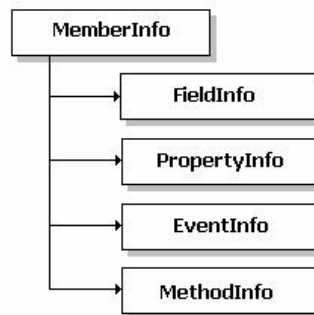
وبالنسبة للعلاقات الوراثية بين الفئات، فيمكنك معرفة ما إذا كان النوع غير قابل للوراثة `NotInheritable` ان كانت قيمة الخاصية `IsSealed` هي `True`، وعلى العكس تعود الخاصية `IsAbstract` بالقيمة `True` ان كان النوع `MustInherit`. نصيحتي لك بان تراجع مكتبة MSDN ان اردت معرفة اكبر قدر من الخصائص.

التعامل مع الأعضاء

في القسم السابق عرضت لك طرق وخصائص يمكنك 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 (حقول، خاصية، حدث، أو طريقة) في النوع، وهي بدورها تعتبر الفئة القاعدية لاربع فئات أخرى هي: `PropertyInfo`، `FieldInfo`، `MethodInfo`، و `EventInfo` (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 نتعامل مع كائنات الفئة القاعدية MemberInfo، فلن نتمكن من إنشائها باستخدام New وإنما 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 نتجنب التكرار نستطيع 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.

§ القيمة BindingFlags.DeclaredOnly تعود بالأعضاء المعرفة في النوع الحالي فقط -أي لا تشمل أعضاء الفئات القاعدية.

فمثلاً، 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 شيئاً 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 للفئات القاعدية.

الطرق والخصائص:

تحتوي الفئة MemberInfo على عشرات الطرق والخصائص التي يمكنك من الاستعلام حول الأعضاء والخاصية بالنوع المرسل، لديك مثلاً 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 يمكنك 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 نتعامل مع كائنات الفئة FieldInfo، فلن نتمكن 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
```

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 الطريقتين 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 نتعامل مع كائنات الفئة PropertyInfo، فلن نتمكن 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 تمكنك 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
```

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 نتعامل مع كائنات الفئة `MethodInfo`، فإن نتمكن من إنشائها باستخدام `New` وإنما 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()`.

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 إن كانت الطريقة مصرحة على أنها MustOverride، والخاصية IsVirtual تعود ايضا بالقيمة True إن 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 نتعامل مع كائنات الفئة EventInfo، فلن نتمكن من انشائها باستخدام New وإنما عليك الحصول على مرجع العضو في النوع، يتم ذلك باستدعاء الطريقة GetEvents() والتي تعود بكافة الأحداث التابعة للنوع (بافتراض أن 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`.

أما الحديث عن الطرق، فالفئة `EventInfo` تحتوي على الطريقة `GetAddMethod()` التي تعود بكائن من النوع `MethodInfo` تمثل الإجراء الذي قام بقتص حدث الكائن باستخدام `AddHandler()`، بينما الطريقة `GetRemoveMethod()` تعود بالإجراء الذي أوقف عملية القنص باستخدام `RemoveHandler()`. أما الطريقة `GetRaiseMethod()` فتعود بكائن من النوع `MethodInfo` أيضا ولكنه يمثل الإجراء الذي يقنص الحدث (أي 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.

إن عادت الخاصية `IsOptional` بالقيمة `True` فهذا يعني أن الوسيطة اختيارية `Optional`، ويمكنك معرفة القيمة الافتراضية للوسيط الاختيارية عن طريق الخاصية `DefaultValue`.

الشفيرة التالية توضح طريقة الاستعلام عن وسيطات الطريقة MySub() والتابعة للفئة

:TestClass

```
Module MainModule

    Class TestClass
        Sub MySub(ByVal x As Integer, ByVal 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 الشفيرة السابقة ستكون شيئاً 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 مثلاً - من اسناد قيمة لخاصية أو استدعاء طريقة.

الفئة ReflectionExample

في هذا القسم سنجري مجموعة كبيرة من التجارب للقراءة والكتابة إلى حقول وخصائص الكائن، بالإضافة 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 الوصول، ...الخ)، وذلك لاختصار وتسهيل الشيفرات البرمجية عليك -و علي أنا ايضا!

إسناد/قراءة قيم الحقول

حاول الحصول على مرجع الحقل باستدعاء الطريقة `GetFields()` أو `GetField()` أو بأي طريقة أخرى، 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 قراءة 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 كوسيلة ثانية لهذه الطريقة، وذلك لان هذه الخاصية لا تحتوي على أية وسائط نرسلها اليها:

```
Dim obj As New ReflectionExample()
Dim P As PropertyInfo = obj.GetType.GetProperty("ExampleProperty")

obj.ExampleProperty = 111
' التحقق من ان الخاصية تم قرانها بشكل صحيح
ArabicConsole.WriteLine(P.GetValue(obj, Nothing)) ' 111
```

في الفقرة التالية استدعاء الطرق سأريك مثالا لارسال وسائط إلى الإجراء، اما الان فلنكمل عملنا مع الخاصية ExampleProperty1 ونحاول إسناد قيمة لها بالطريقة SetValue() (لا ننسى ارسال القيمة Nothing مع الوسيلة 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 استدعاؤها بالطريقة Invoke() والتي تتطلب وسائط الطريقة، ان كانت الطريقة لا تحتوي على

أية وسيطات فأرسل القيمة Nothing، اما إن وجدت فأرسل مصفوفة من النوع Object عدد 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 كان للطريقة وسيطات اختيارية Optional Parameters، فأرسل القيمة Type.Missing في المصفوفة المراد إرسالها كقيم لوسيطات الطريقة، في الشيفرة التالية قمنا بعمل استدعائين للطريقة ExampleMethod2()، الأول لا يرسل قيمة للوسيط الاختيارية، 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 اختتم 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 إمكانية إنشاء الكائنات لحظة التنفيذ بذكر الاسم النصي للكائن والذي تكون صيغته مبدئية 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 أخرى يمكنك من إنشاء الكائن ديناميكياً 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
```

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 يمكنك هذه الفئة من معرفة ذلك؟ والجواب بفضل مجموعة من الفئات في مجال الأسماء System.Diagnostics -تصنف من فئات الانعكاس Reflection أيضا.

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 MethodInfo (يتبع إلى مجال الأسماء System.Collection واستخدامه شبيه بـ 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 أول إجراء يدخل طاير الاستدعاءات:

```

** CallerSub *****
CallerSub ()
main ()

** CalleeSub *****
CalleeSub ()
CallerSub ()
main ()

```

أسلوب أفضل:

بدلاً من كتابة شيفرات الحلقة التكرارية في كل مرة تود معرفة طاير الاستدعاءات، لما لا نعرف إجراء خاص بك بالإسم GetStack() يعود بمصفوفة حرفية من النوع String تمثل الإجراءات في طاير 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.

الجزء الثالث

تطوير تطبيقات Windows

نماذج Windows Forms

ان كنت تنوي تطوير تطبيقات تحاكي تطبيقات Windows القياسية، فعليك إظهار مخرجاتك على نوافذ عوضا عن الكائن ArabicConsole، حيث ستحتاج إلى استخدام مجموعة كبيرة من الفئات مشمولة في مجال الأسماء System.Windows.Forms. يحتوي الجزء الثالث **تطوير تطبيقات Windows** من هذا الكتاب على أربعة فصول تختص بتطوير مشاريع من نوع Windows Application. الأول مدخلك لاستخدام نماذج Windows Forms، الثاني يتمحور حول الأدوات Controls التي تحضنها النماذج، الثالث موجه لاستخدام تقنية GDI+، اما الرابع فيلقي الضوء على مجموعة من المواضيع المتفرقة والتي يتعامل معها مطورو تطبيقات Windows بشكل مكثف.

ملاحظة

إن ما زلت مستمر على المشاريع من النوع Console Application، فعليك استيراد مجال الاسماء التالي:

```
Imports System.Windows.Forms
```

اما ان كنت متابع لهذا الفصل، فليست بحاجة لاستيراده حيث ان المشاريع من نوع Windows Application تقوم باستيراده بشكل تلقائي في خانة التبويب Imports من صندوق الحوار Project Property Pages (شكل 2-7).

مدخلك إلى نماذج Windows Forms

في Visual Basic .NET، نافذة النموذج ما هي إلا فئة تقليدية مشتقة وراثياً من الفئة `System.Windows.Forms.Form` تصرحها في أي مكان من البرنامج كما تصرح الفئات الأخرى:

```
Public Class Form1
    Inherits System.Windows.Forms.Form
    ...
    ...
End Class
```

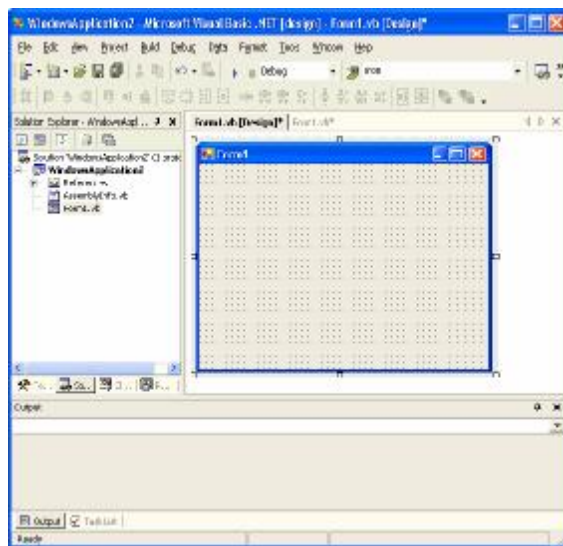
بعد تصريحك لهذه الفئة، يمكنك إنشاء كائن منها واستدعاء الطريقة `Show()` لإظهار النافذة:

```
Dim myForm As New Form1
myForm.Show ( )
```

مع ذلك، تتطلب فئات النماذج مجموعة إضافية من الشيفرات المصدرية عليك اضافتها بين فكي التركيب `Class ... End Class` قبل إنشاء الكائنات منها. مع أنه يمكنك كتابة هذه الشيفرات يدوياً بنفسك، إلا أنني لا أجد سبب مقنعاً يمنعك من استخدام مصمم النماذج `Form Designer`.

مصمم النماذج Form Designer

يصنف مصمم النماذج من برامج مولدة الشيفرات `Code Generator`، حيث يقوم بتوليد الشيفرات التي توافق التصميم التي تتجزأ بنقرات بسيطة بالفأرة (شكل 1-13). يمكن للملف الواحد أن يحتوي على أكثر من فئة نموذج `Form Class`، ولكن الشيفرات المولدة من مصمم النماذج ستوجه إلى الفئة المسطورة في أعلى الملف فقط.



شكل 13-1: مصمم النماذج Form Designer.

ملاحظة

لا يشترط ان تكون فئة النموذج في أعلى الملف فقط لاستخدام مصمم النماذج، بل حتى لا يمكن للفئة ان تكون محصورة في فئة اخرى أو وحدة برمجية Module. نستنتج من هذه الملاحظة أيضا، ان محدد الوصول Private لا يمكن استخدامه مع فئات النماذج لاستخدام مصمم النماذج.

أنشئ مشروع جديد من نوع Windows Application، ستلاحظ ان بيئة التطوير Visual Studio .NET قد أنشأت ملفات المشروع وأظهرت لك نافذة خالية بعنوان Form1. انقر المزدوج على هذه النافذة يفتح لك نافذة محرر الشيفرة Code Editor، ستري ان مصمم النماذج قد ولد الشيفرة التالية بشكل ابتدائي (تعمدت ترجمة التعليقات إلى اللغة العربية):



```
Public Class Form1
    Inherits System.Windows.Forms.Form

#Region " Windows Form Designer generated code "

    Public Sub New()
        MyBase.New()

        ' هذا الاستعداد مطلوب من مصمم النماذج
        InitializeComponent()

        ' اصف أي شيفرات اضافية لهذا المشيد بعد هذا السطر

    End Sub

    ' مهدم الفئة.
    Protected Overloads Overrides Sub Dispose(ByVal disposing As
Boolean)
        If disposing Then
            If Not (components Is Nothing) Then
                components.Dispose()
            End If
        End If
        MyBase.Dispose(disposing)
    End Sub

    ' متغير خاص بمصمم النماذج
    Private components As System.ComponentModel.IContainer

    ' الاجراء التالي خاص بمصمم النماذج
    ' يمكنك تعديل محتوياته من خلال نافذة المصمم فقط
    ' ولا تحاول تعديل الشيفرة يدويا
    <System.Diagnostics.DebuggerStepThrough()> Private Sub
InitializeComponent()
        '
        'Form1
        '
        Me.AutoScaleBaseSize = New System.Drawing.Size(5, 13)
        Me.ClientSize = New System.Drawing.Size(292, 269)
        Me.Name = "Form1"
        Me.Text = "Form1"

    End Sub

#End Region
End Class
```

قم بقنص الحدث Click (الموجود في الفئة القاعدية Base Class) والذي يتم تفجيره لحظة نقر المستخدم على النافذة، وأضف شيفرة تبين لنا ردة الفعل. عرفت هنا إجراء باسم formWasClicked لقنص الحدث Click يعرض رسالة ترحيبية باستخدام الدالة MsgBox:



```
Public Class Form1
    Inherits System.Windows.Forms.Form

    ...

    Private Sub formWasClicked(ByVal sender As Object, _
        ByVal e As System.EventArgs) Handles MyBase.Click

        MsgBox("Windows Form في مرحبا بك في")
    End Sub
End Class
```

انظر أيضا

لمزيد من التفاصيل حول الأحداث وقنصها، راجع الفصل الثالث **الغنائ والكائنات**.

بعد تنفيذك للبرنامج ستظهر لك نافذة النموذج، وان قمت بالنقر عليها (بزر الفأرة الأيسر أو الأيمن) في أي مكان، ستظهر لك الرسالة الترحيبية (شكل 13-2):



شكل 13-2: الرسالة الترحيبية ظهرت نتيجة للنقر على النافذة.

نظرة حول الشيفرة المولدة

استيعابك للشيفرة التي ولدها مصمم النماذج يعتمد اعتماد كلي على استيعابك للفصول الخمس الأولى من هذا الكتاب، فلا يوجد شيء جديد بها سوى الشيفرات المتعلقة بالمتغير components والخاص بمصمم النماذج فقط وليس له أي تأثير كبير في الكائن المنشئ من هذه الفئة لحظة التنفيذ.

في أعلى الشيفرة عرفنا فئة باسم Form1 مشتقة وراثياً من الفئة System.Windows.Forms.Form -وهي احد متطلبات فئات النماذج:

```
Public Class Form1
    Inherits System.Windows.Forms.Form
    ...
    ...
End Class
```

بالنسبة لمشييد الفئة، فعليه استدعاء مشيد الفئة القاعدية واستدعاء الإجراء InitializeComponent() والخاص بمصمم النماذج اولاً، ومن ثم يمكنك اضافة أي شيفرات تود تنفيذها لحظة إنشاء نسخة جديدة من كائن:

```
Public Class Form1
    ...
    ...
    Public Sub New()
        MyBase.New()

        ' هذا الاستدعاء مطلوب من مصمم النماذج '
        InitializeComponent()

        ' اصف أي شيفرات اضافية لهذا المشيد بعد هذا السطر '
    End Sub
    ...
    ...
End Class
```

بعد ذلك، نقوم بإعادة تعريف Overloads وإعادة قيادة Overrides الطريقة Dispose() من الفئة القاعدية، ليكون المهدم Destructor الخاص بالفئة (من الضروري استدعاء مهدم الفئة القاعدية):

```
Public Class Form1
    ...
    Protected Overloads Overrides Sub Dispose(ByVal _
        disposing As Boolean)

        If disposing Then
            If Not (components Is Nothing) Then
                components.Dispose()
            End If
        End If
        MyBase.Dispose(disposing)
    End Sub
    ...
End Class
```

اخيراً، الإجراء `InitializeComponent()`، وهو خاص بمصمم النماذج، حيث يتم فيه توليد الشيفرة الناتجة عن التعديلات التي تفعلها بمصمم النماذج (كتغيير الخصائص، إضافة الأدوات، وغيرها) وقت التصميم.

```
Public Class Form1
    ...
    ...
    ' الإجراء التالي خاص بمصمم النماذج
    ' يمكنك تعديل محتوياته من خلال نافذة المصمم فقط
    ' ولا تحاول تعديل الشيفرة يدوياً
    <System.Diagnostics.DebuggerStepThrough()> _
    Private Sub InitializeComponent()
        '
        'Form1
        '
        Me.AutoScaleBaseSize = New System.Drawing.Size(5, 13)
        Me.ClientSize = New System.Drawing.Size(292, 269)
        Me.Name = "Form1"
        Me.Text = "Form1"
    End Sub
End Class
```

ملاحظة

لا تحاول تعديل الشيفرة المصدرية والمولدة في الإجراء `InitializeComponent` يدوياً بنفسك، وإنما اعتمد على مصمم النماذج فهو يقوم باللازم نيابة عنك بدقة أكثر.

لست مضطر لإضافة جميع الشيفرات السابقة في كل مرة تعرف فيها فئة نموذج جديدة، إذ يمكنك حذف الشيفرات الخاصة بمصمم النماذج وتكتب الشيفرات الضرورية لهذه الفئات فقط (انصحك بشدة بعدم عمل ذلك):

```
Public Class Form1
    Inherits System.Windows.Forms.Form

    Public Sub New()
        MyBase.New()
    End Sub
```

```

Protected Overloads Overrides Sub Dispose(ByVal disposing As Boolean)
    MyBase.Dispose(disposing)
End Sub
End Class

```

التعامل مع أكثر من نموذج

بادئ ذي بدء، عليك معرفة ان لكل مشروع من النوع Windows Application نافذة رئيسية تسمى النافذة الابتدائية **Startup Window** يمكنك تحديدها عند خانة Startup Object في نافذة Project Property Pages (شكل 13-3). هذه النافذة لها طابع خاص عن سائر نوافذ البرنامج، فهي ستظهر مع تنفيذ البرنامج بشكل تلقائي دون الحاجة لتعريف كائن من فئتها واستدعاء الطريقة Show(). شيء اخر مهم حول هذه النافذة، وهو ان نهاية البرنامج تعتمد على إغلاق المستخدم لهذه النافذة، وحتى لو وجدت عشرات النوافذ المفتوحة سيتم إغلاقها أيضا.



شكل 13-3: تحديد النافذة الابتدائية للمشروع.

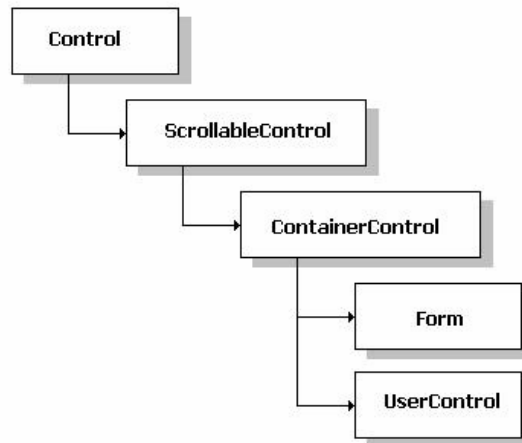
مع ذلك، يمكنك تحديد النافذة الابتدائية وقت التنفيذ في كل مرة تستدعي فيها الإجراء Application.Run() والذي تتطلب وسيطته الوحيدة كائن النافذة، أفضل مكان يمكنك استخدام الطريقة Run() السابقة هو الإجراء الابتدائي Sub Main() حيث تحدد من البداية النافذة الابتدائية:

```
Module MainModule
    Sub main()
        Dim frmMain As New Form1()

        Application.Run(frmMain)
    End Sub
End Module
```

محل الفئة Form من الإعراب

محل الفئة Form من الإعراب في فضاء الاسماء System.Windows.Forms يوضحه لك (شكل 13-4)، حيث يعرض لك العلاقة الوراثية بين الفئة Form ومجموعة من الفئات في فضاء الاسماء System.Windows.Forms.



شكل 13-4: العلاقة الوراثية بين الفئة Form ومجموعة من الفئات الأخرى.

الفئة القاعدية Control تمثل أداة Control تقليدية تظهر في صندوق الأدوات Toolbox (شكل 14-1 بالفصل القادم) يمكنك وضعها على نافذة النموذج. والفئة ScrollableControl تمثل أيضا أداة Control تقليدية ولكن لها سمات إضافية تتمحور حول دعم أشرطة التمرير Scroll Bars للأداة. بالنسبة للفئة ContainerControl فهي مشتقة من الفئة ScrollableControl تضيف إليها قابلية ان تكون الاداة حاضنة Container لأدوات أخرى.

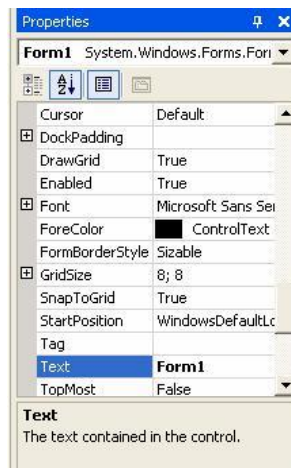
سأحدث عن جميع الفئات الموجودة في (الشكل 13-4) السابق بالتفصيل الممل في الفصل القادم **الأبواب Controls**، حيث كان هدفي من هذه الفقرة فقط توضيح الفئات القاعدية للفئة Form واعلامك ان كل الخصائص، الطرق، والاحداث التابعة لهذه الفئات (والتي لن أتطرق لها في هذا الفصل) موجودة أيضا في الفئة Form.

الخصائص، الطرق، والأحداث

كما ذكرت قبل قليل ان معظم الخصائص، الطرق، والأحداث التابعة للفئة Form هي مشتقة من فئات أخرى أجلت تفصيلها إلى الفصل القادم. لذلك، سأعرض لك في هذا القسم أعضاء الفئة Form فقط بشكل مختصر، وبإمكانك مراجعة مكتبة MSDN للحصول على أكبر قدر من التفاصيل.

خصائص النموذج

قبل ان ابدأ في عرض خصائص النموذج، بودي اخبارك انك تستطيع تعديل قيمها وقت التصميم عن طريق نافذة الخصائص Properties (شكل 13-5) والتي تصل لها بالضغط على المفتاح [F4] أو اختيار الامر Properties Windows من قائمة View.



شكل 13-5: نافذة الخصائص Properties.

العمود الايسر يمثل اسماء الخصائص بينما الأيمن قيمة كل خاصية من هذه الخصائص، تذكر ان أي تعديل على هذه الخصائص يقوم بتوليد الشيفرة المناسبة في الإجراء InitializeComponent() التابع لمصمم النماذج Form Designer في فئة النافذة. جرب - مثلا- تغيير قيمة الخاصية Text، ستلاحظ ان مصمم النماذج أضاف هذا السطر:

```
Private Sub InitializeComponent()  
    ...  
    ...  
    Me.Text = "النافذة الرئيسية"  
End Sub
```

خصائص المظهر:

للتحكم في شريط النافذة العلوي، لديك الخاصية Text السابقة والتي كان غرضها تعديل العنوان الذي يظهر في شريط النافذة العلوي وهي خاصة حرفية من النوع String. بينما الخاصيتان MaximizeBox و MinimizeBox منطقية من النوع Boolean حيث تحدد فيها إمكانية استخدام زر التكبير والتصغير الخاص بالنافذة، ان جعلت قيم كلا الخاصيتين False فستختفي الأزرار من شريط النافذة، كما سيختفي صندوق التحكم وزر الإغلاق أيضا ان أسندت القيمة False للخاصية ControlBox.

الخاصية Icon فتحدد فيها رمز (أيقونة) للنافذة، والخاصية BackgroundImage يمكنك من وضع صورة تغطي سطح النافذة، ضع في اعتبارك بان الصورة ستكرر حتى تغطي كامل سطح النافذة (كتأثير الاختيار Tile الذي تحدده عن تحديد صورة لخلفية سطح المكتب Wall Paper).

الخاصية Opacity تسند لها قيمة مجالها من 1 إلى 0 تحدد فيها مقدار شفافية النافذة، القيمة 1 تظهر النافذة دون شفافية، والقيمة 0 تخفي النافذة تماما، جرب وضع القيمة 0.5 لتصبح النافذة شبه شفافة.

بالنسبة للخاصية TransparencyKey، ففيها تحدد اللون الذي تريد إخفائه من النافذة لحظة التنفيذ. تذكر ان اخفاء اللون من النافذة يعطي فرصة كبيرة لظهور النوافذ التي خلف النافذة الحالية، بل ويمكن أيضا المستخدم من اختراق النافذة ليصل إلى النوافذ التي خلفها. من الخصائص أيضا، الخاصية ShowInTaskBar والتي تظهر زر النافذة في شريط المهام Task Bar والخاص بنظام التشغيل.

يمكنك اختيار حد من 7 حدود تدعمه الخاصية `FormBorderStyle`: القيمة `Sizable` تضمن للمستخدم قدرته على تحجيم النافذة، اما القيمتين `FixedSingle` و `Fixed3D` فتمنع المستخدم من تحجيم النافذة، كما تفعل القيمة `None` والتي تلغي الحدود بشكل نهائي. اخيرا، يمكنك اسناد القيمة `Hide` للخاصية `SizeGripStyle` ان اردت اخفاء عصا التحجيم `Size Grip` للنافذة (عصا التحجيم هي خطين يظهران في الزاوية السفلى اليمنى للنافذة).

خصائص الموقع والحجم:

بالنسبة للخاصيتين `DesktopBounds` و `DesktopLocation` فتوجد نسختين منهما معرفة كطرق هما `SetDesktopBounds` و `SetDesktopLocation` سأعود اليهما لاحقا في فقرة طرق النموذج.

يمكنك جعل نافذة النموذج تظهر في وسط الشاشة باسناد القيمة `CenterScreen` إلى الخاصية `StartPosition`. كما تستطيع تحديد الحجم الابتدائي للنافذة عن طريقة الخاصية `WindowState` والتي يمكن ان تكون مكبرة `Maximized`، مصغرة `Minimized`، أو الحجم الطبيعي `Normal`.

وعلى ذكر حجم النافذة، يمكنك أيضا تحديد مقاييس لحجم التكبير والتصغير للنافذة عن طريق الخاصيتين `MaximumSize` و `MinimumSize`، رغم اني انصحك بعدم تغيير هاتين القيمتين حتى لا تسبب إرباك لمستخدمي نوافذك والذين اعتادوا على ان يكون الحجم الكبير يغطي الشاشة والصغير يظهر شريط العنوان فقط للنافذة.

اخيرا، اسند القيمة `True` إلى الخاصية `TopMost` ان اردت جعل النافذة فوق النوافذ الاخرى لكافة تطبيقات Windows.

خصائص أشرطة التمرير Scrolling:

كل ما هو مطلوب منك إسناد القيمة `True` إلى الخاصية `AutoScroll` ان اردت اضافة اشطرة تمرير `Scroll Bars` إلى نافذة النموذج، ضع في اعتبارك ان اشطرة التمرير لن تظهر إلا ان كان حجم النافذة لا يغطي كافة الأدوات المحضونة بها. يمكنك الاستعلام وقت التنفيذ ما اذا كان شريط التمرير الافقي أو العمودي قد ظهرت فعلا عن طريق الخاصيتين `HScroll` و `VScroll`.

الخاصية `AutoScrollMargin` تحدد فيها حجم الحدود الوهمية التي تحيط بالأداة، هذه الحدود الوهمية ليست ظاهرة، وانما تستخدمها نافذة النموذج لبدء عرض اشطرة التمرير ان وصلت حدود النموذج هذه الحدود الوهمية. يفضل إبقائها كما هي (0; 0).

تستطيع الاستعلام ومعرفة مواقع أشرطة التمرير الحالية عن طريق الخاصية `AutoScrollPosition` والتي تعود بكائن من النوع `Point`. يمكنك أيضا اسناد قيم لها بإنشاء كائن من النوع `Point` لتحريك أشرطة التمرير برمجيا، الإحداثيات التي ترسلها تمثل النقطة الظاهرة في الزاوية العلوية اليسرى للنافذة:

```
' حرك اشرطة التمرير بحيث تمثل تظهر النقطة (10، 20) في '
' الزاوية العليا اليسرى من النافذة '
Me.AutoScrollPosition = New Point(10, 20)
```

انظر أيضا

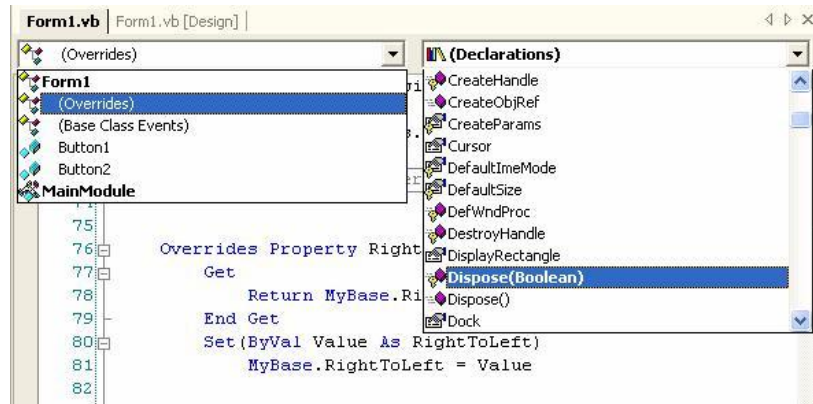
لي عودة حول الكائنات من النوع `Point` في الفصل القادم الأدوات `.Controls`.

إلى جانب الخاصية `AutoScrollPosition`، توجد الطريقة `ScrollControlIntoView()` والتي تحرك أشرطة التمرير بحيث تظهر لك الاداة التي ترسل كوسيلة لها:

```
Me.ScrollControlIntoView (TextBox1)
```

طرق النموذج

قبل عرض الطرق الخاصة بنافذة النموذج، بودي تذكيرك بأن معظم الطرق (والخصائص أيضا) قابلة لإعادة القيادة `Overrides` (فئات النماذج مشتقة من الفئة `Form`)، ويمكنك الاستعانة بمحرر الشيفرة لإعادة قيادتها باختيار `Overrides` من القائمة العلوية اليسرى ومن ثم اسم الطريقة أو الخاصية من القائمة العلوية اليمنى (شكل 13-6 بالصفحة التالية):



شكل 13-6: الاستعانة بمحرر الشيفرة لإعادة قيادة الطرق والخصائص.

يمكنك اظهار نافذة النموذج وقت التنفيذ -كما ذكرت- باستدعاء الطريقة Show() للكائن المنشئ منها، ولكن ضع في عين الاعتبار ان الشيفرة التي تلي استدعاء الطريقة Show() سيتم الاستمرار في عملية تنفيذها أيضا:

```
Dim MyForm As New Form2
```

```
MyForm.Show ( )
```

الشيفرة التالية سيتم تنفيذها أيضا :

```
...  
...
```

السبب في استمرار تنفيذ الشيفرة التي تلي الاستدعاء للطريقة Show() يتعلق بالبنية التحتية لتركيبية نوافذ نظام التشغيل Windows، حيث ان الاستدعاء Show() يظهر نافذة تسمى -في عالم برمجة Windows- بالـ **Modeless Window** وهو اسلوب يمكن الشيفرة المستدعية للنافذة من الاستمرار في التنفيذ، كما يمكن مستخدم النافذة أيضا من العودة للنافذة وتنشيطها. اما استدعائك للطريقة ShowDialog() فهو مناسب جدا لفتح نوافذ من النوع **Modal Window** (اغلب صناديق الحوار Dialog Boxes يتم فتحها بهذا الاسلوب) حيث توقف عمل النافذة الحالية ولن يتمكن المستخدم من العودة إلى النافذة الفاتحة لها حتى يغلق النافذة المفتوحة:

```
Dim MyForm As New Form2
```

```
MyForm.ShowDialog()
```

لن يتم تنفيذ الشيفرة التالية حتى ' يغلق المستخدم النافذة '

```
...  
...
```

بعد فتح النافذة بالطريقة Show() ستصبح هي النافذة النشطة Window Active في البرنامج بشكل تلقائي، مع ذلك يمكن تغيير النافذة النشطة في البرنامج باستخدام الطريقة Activate().

يمكنك في أي وقت اخفاء النافذة باستدعاء الطريقة Hide() والتي تخفي النافذة من عين المستخدم فقط، حيث أنها لا تزال على قيد الحياة، يمكنك إعادة عرضها مرة أخرى باستدعاء الطريقة Show() أيضا. اما ان رغبت في اغلاق النافذة بشكل نهائي، فالطريقة Close() ستكون وافية وكافية.

الطريقة SetDesktopLocation() تمكنك من تحريك النافذة وتغيير موقعها على سطح المكتب، الوسيطة الاولى تمثل المحور السيني x والثانية المحور الصادي y، حيث تمثل النقطة (0, 0) الزاوية العليا اليسرى من سطح المكتب، تزيد أفقيا كلما اتجهنا يمينا، وعموديا كلما اتجهنا إلى الأسفل. وان رغبت في تحريك النافذة مع تغيير حجمها في ضربة واحدة، استدعي الطريقة SetDesktopBounds() والتي تتطلب وسيطتين إضافيتين هما عرض النافذة وارتفاعها:

```
Dim MyForm As New Form2
```

```
MyForm.SetDesktopBounds(0, 0, 200, 100)
```

الطريقتين AddOwnedForm() و RemoveOwnedForm():

في بيئة Windows توجد فلسفة النوافذ المملوكة **Owner Windows** والنوافذ المملوكة **Owned Windows**. النوافذ المملوكة تظهر دائما فوق النافذة المملوكة لها، ويتم إغلاقها بشكل تلقائي ان أغلقت النافذة المملوكة. المزيد أيضا، عند تصغير النافذة المملوكة Minimize سيتم تصغير كافة النوافذ المملوكة وتظهر جميعها في زر واحد في أسفل شريط المهام Task Bar. يمكنك لنوافذك ان تمتلك نوافذ أخرى بإضافتها بالطريقة AddOwnedForm():

```
Dim x As New ownedForm()  
Dim y As New ownedForm()
```

```
Me.AddOwnedForm(x)
Me.AddOwnedForm(y)

x.Show()
y.Show()
```

بعد اضافة النافذة كمملوكة، يمكنك الاستعلام عنها بالخاصية `OwnedForms` والتي تعود بمرجع يمثل جميع النوافذ المملوكة:

```
Dim frm As Form

For Each frm In Me.OwnedForms
    Frm.Text = "نافذة مملوكة"
Next
```

يمكنك معرفة ما اذا كانت النافذة مملوكة لنافذة اخرى عن طريق الخاصية `Owner` والتي تعود بمرجع لنافذة النموذج المالكة:

```
If Me.Owner Is Nothing
    ' النافذة الحالية لا تمتلكها أي نافذة اخرى
    ...
Else
    ' النافذة الحالية مملوكة من قبل نافذة اخرى
    ...
End If
```

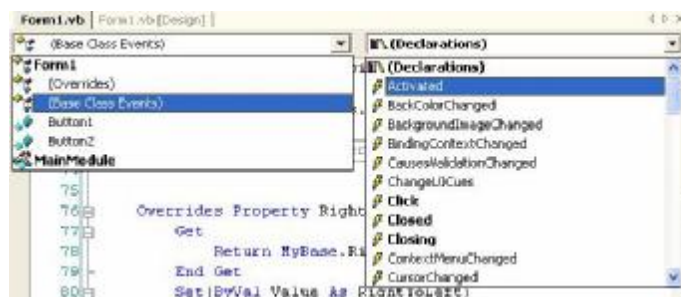
اخيرا، تستطيع الاستغناء عن ملكية النافذة باستدعاء الطريقة `RemoveOwnedForm()` والتي تتطلب منك وسيطة واحد تمثل مرجع النافذة المملوكة.

أحداث النموذج

نافذة النموذج هي أكثر كائن من كائنات إطار عمل `NET Framework`. بشكل عام يحتوي على أحداث، اغلب أحداثه مشتقة وراثيا من الفئة `Control` ولذلك سأؤجل تفصيل هذه الأحداث حتى الفصل القادم، اما هنا فسأعرض الأحداث الخاصة بنافذة النموذج فقط.

قبل البدء بعرض الأحداث الخاصة بنافذة النموذج، دعني أذكرك بأن محرر الشيفرات والخاص ببيئة التطوير `Visual Studio .NET` يسهل عليك امر قنص الأحداث، وذلك باختيار

Base Class Events من القائمة العلوية اليسرى، ومن ثم اختيار الحدث المراد قنصه من القائمة العلوية اليمنى (شكل 13-7).



شكل 13-7: الاستعانة بمحرر الشيفرات لقنص الأحداث.

يمكنني ان الخص لك تسلسل الأحداث التي تقع على نافذة النموذج من بداية إنشاء كائنه وفتح النافذة حتى اغلاق النافذة وقتل كائنه بهذا التسلسل:

المشيد New() -> الحدث Load -> الحدث Paint -> الحدث Activated -> الحدث Deactivated -> الحدث Closing -> الحدث Closing -> الحدث Closed -> المهدم Dispose().

ملاحظة

توجد أحداث أخرى لم أتطرق لها داخله في عناصر السلسلة السابقة (ك Move, Resize, LostFocus, GotFocus... الخ) فضلت تأجيلها إلى الفصل القادم، وذلك لأنها تتبع للفئة القاعدية Control بالأصل وليس Form.

المشيد New() والمههم Dispose():

تحدثت ما فيه الكفاية عن المشيدات والمهدمات سابقا في الفصل الثالث **الفئات والكائنات**، ولن اضيف شيئا جديدا هنا إلا تذكيرك باستدعاء المشيد والمههم للفئة القاعدية Base Class قبل كتابة حرف واحد من حروف الشيفرات المصدرية:

```

Public Sub New()
    MyBase.New()
    ...
    ...
End Sub

Protected Overloads Overrides Sub Dispose(ByVal _
    disposing As Boolean)

    If disposing Then
        ...
        ...
    End If

    MyBase.Dispose(disposing)
    ...
    ...
End Sub

```

انظر أيضا

يمكنك مراجعة الفقرة الفرعية **أسلوب أكثر أمانا لكتابة المهدمات** في الفصل الثالث **الفئات والكائنات** لاستيعاب المهدم `Dispose()`. كما تحدثت عن محدد الوصول `Protected` وإعادة التعريف `Overloading` وإعادة القيادة `Overriding` في الفصل الرابع **الوراثة**.

الحدث Load:

يتم تنفيذ هذا الحدث بمجرد البدء في تحميل النافذة وقيل إكمال ظهورها. عند استدعائك للطريقة `Show()` التي تظهر النافذة، سيتم تنفيذ هذا الحدث مرة واحدة فقط ولكن قبل أن تظهر النافذة، وللتأكد من ذلك جرب كتابة هذه الشيفرة في إجراء فنص الحدث `Load`:

```

Private Sub Form1_Load(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles MyBase.Load

    MsgBox("جاري تحميل النافذة")
End Sub

```

عند فتح النافذة بالطريقة `Show()`، ستلاحظ أن الرسالة السابقة ستظهر قبل ظهور النافذة.

الحدث Paint:

يتم تنفيذ الحدث Paint كلما دعت الحاجة إلى إعادة رسم النافذة، فلو وضعت النافذة س فوق النافذة ص ومن ثم حركت النافذة س لتظهر النافذة ص، سيتم تنفيذ الحدث Paint التابع للنافذة ص. كذلك، عند ظهور أي جزء مخفي من النافذة نتيجة تحريكها خارج حدود سطح المكتب سيتم تقجير الحدث Paint أيضا.

عند ظهور أو اختفاء اشطرة التمرير Scrollbars والتابعة للنافذة، سيتم تنفيذ الحدث Paint أيضا، كذلك الحال عند تحريك اشطرة التمرير من قبل المستخدم فهو سبب بديهي لاعادة رسم النافذة.

من الحالات الاخرى التي تؤدي إلى تنفيذ الحدث Paint تكبير حجم النافذة من قبل المستخدم، وذلك لظهور اجزاء اضافية تتطلب إعادة الرسم، اما تصغير حجم النافذة فلا يؤدي إلى تنفيذ الحدث، وان كان لديك سبب وجيه لتنفيذ الحدث Paint عند تصغير حجم النافذة، فيمكنك استدعائه من داخل الحدث:

```
Private Sub Form1_Resize(ByVal sender As Object, _
    ByVal e As System.EventArgs) Handles MyBase.Resize

    Me.Form1_Paint(Me, Nothing)
End Sub
```

الاستدعاء السابق صحيح ولكنه يطلق الحدث Paint مرتين الاول بسبب إعادة الرسم والثانية بسبب الاستدعاء الناتج من تغيير الحجم، لذلك يفضل استدعاء الطريقة Refresh() والتي تعيد رسم النافذة كلما دعت الحاجة:

```
Private Sub Form1_Resize(ByVal sender As Object, _
    ByVal e As System.EventArgs) Handles MyBase.Resize

    Me.Refresh()
End Sub
```

الحداث Activated و Deactivated:

في كل مرة تصبح النافذة الحالية هي النافذة النشطة Active Window، سيتم تنفيذ الحدث Activated، وبمجرد اختفاء التركيز عن النافذة سيتم تنفيذ الحدث Deactivated.

الحدثان Closing و Closed:

يتم تنفيذ الحدث Closing بمجرد البدء باغلاق النافذة سواء برمجيا باستدعاء الطريقة Close() أو ان قام المستخدم بالضغط على زر اغلاق النافذة. مع ذلك، لديك فرصة كبيرة في منع عملية الاغلاق وذلك بارسال القيم True إلى الخاصية Cancel التابعة لوسيلة الحدث:

```
Private Sub Form1_Closing(ByVal sender As Object, ByVal e _
    As ComponentModel.CancelEventArgs) Handles MyBase.Closing

    e.Cancel = True
End Sub
```

ان تم اغلاق النافذة بشكل نهائي واختفت من عين المستخدم، سيأتي دور تنفيذ الحدث Closed معطيا لك فرصة اخيرة لعمل ما تريد، مع العلم ان هذا الحدث يعني ان النافذة قد تم اغلاقها ولن تتمكن من الغاء عملية الاغلاق أو حتى اعادة اظهار النافذة باستدعاء الطريقة Me.Show().

أحداث أخرى:

من الأحداث الاخرى والخاصة بنافذة النموذج الحدث MinimumSizeChange الذي يتم تنفيذه بمجرد تغيير قيمة الخاصية MinimumSize، الحدث MaximumSizeChange يتم تنفيذه بمجرد تغيير قيمة الخاصية MaximumSize، والحدث MaximizedBoundsChange المرافق للخاصية MaximizedBounds.

نماذج MDI Forms

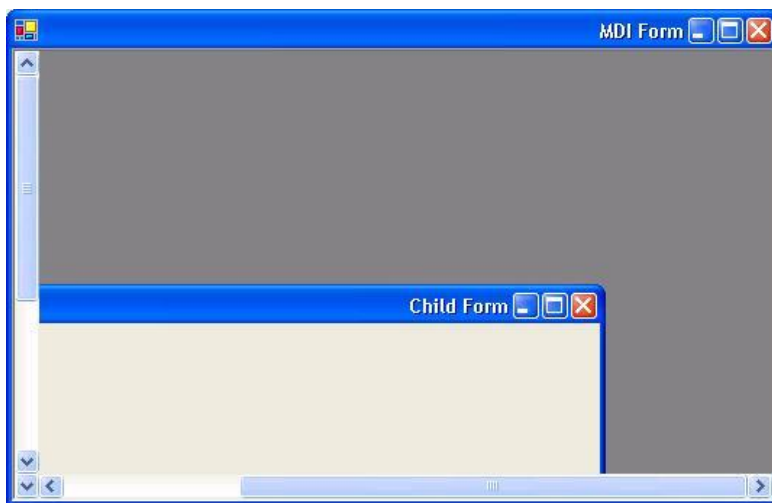
يمكن لأي نافذة نموذج سمها كان اصلها وفصلها - ان تكون نافذة من النوع MDI، وهو النوع الذي يمكن النافذة من ان تحضن نوافذ اخرى. كل ما هو مطلوب منك لجعل النافذة من النوع MDI اسناد القيمة True إلى الخاصية IsMdiContainer والتي يمكن ان تكتب وتقرأ وقت التصميم والتنفيذ.

من المزايا في نماذج Windows Forms هو امكانية تعريف أكثر من نافذة MDI في برنامج واحد. مع ذلك، لا تحاول عمل ذلك إلا ان دعت الحاجة فعلا لذلك، فالأغلبية الساحقة لتطبيقات Windows من النوع MDI تستخدم نافذة MDI واحدة.

النوافذ الأبناء Child Windows

كما يمكن لأي نافذة -سهما كان أصلها وفصلها- أن تكون نافذة من النوع MDI، يمكن لأي نافذة أيضاً أن تكون نافذة ابنة (النوافذ المحصورة في نوافذ MDI تسمى **نوافذ أبناء Child**)، اسند مرجع إلى النافذة MDI في الخاصية MdiParent التابعة للنافذة الابنة (شكل 13-8):

```
Sub main()  
    Dim frmMain As New Form1()  
    Dim frmChild As New Form1()  
  
    frmMain.Text = "MDI Form"  
    frmMain.IsMdiContainer = True  
  
    frmChild.MdiParent = frmMain  
    frmChild.Text = "Child Form"  
    frmChild.Show()  
  
    Application.Run(frmMain)  
End Sub
```



شكل 13-8: نافذة ابنة Child محصورة داخل نافذة MDI.

ضع في عين الاعتبار، أن النافذة الابن تكون مملوكة للنافذة MDI المحددة في الخاصية MdiParent، نستنتج من ذلك، أن النافذة الابن سيتم إغلاقها تلقائياً إن أغلقت النافذة المالكة، كما سيتم تصغيرها تلقائياً إن تم تصغير النافذة المالكة.

الفرق البسيط بين ملكية النوافذ باستخدام الطريقة AddOwnedForm(), و ملكية النوافذ بإسناد قيمة للخاصية MdiParent هو ان الثانية تكون النوافذ المملوكة محصورة داخل حدود النافذة المالكة، بينما في الحالة الاولى فتكون النوافذ المملوكة خارج حدود النافذة المالكة. فرق اخر يتعلق بطريقة اظهار القوائم Menus، حيث ان قوائم النوافذ المملوكة تعرض في نفس نافذة MDI (النافذة المالكة)، بينما في حالة الملكية باستخدام الطريقة AddOwnedForm() فكل قائمة تعرض في نافذتها بشكل مستقل.

ملاحظة

عند تكبير النافذة الابن Maximize وهي في داخل النافذة MDI، سيتم تغيير العنوان الطاهر في أعلى النافذة MDI بحيث يشمل عنوان نافذة الابن المكبرة والنافذة الام.

خصائص وطرق إضافية

من الطرق والخصائص التي تتعلق بالنوافذ الابناء Child Windows الخاصية IsMdiChild التي تعود بالقيمة True ان كانت النافذة ابنة:

```
If Me.IsMdiChild Then
    ...
End If
```

مع ذلك، انصحك بالعودة إلى الخاصية MdiParent حيث انها تعود بمرجع للنافذة MDI -ان وجدت- عوضا عن القيمة True، مما يمكنك من الوصول إلى كافة طرق وخصائص النافذة الحاضرة:

```
If Not Me.MdiParent Is Nothing Then
    Me.MdiParent.Text = "...
    ...
End If
```

وفي سياق نافذة MDI الام، فيمكنك معرفة جميع النوافذ المحصورة بها عن طريق الخاصية MidiChildren، والتي تعود بمصفوفة تمثل مراجع إلى جميع نوافذ الابناء:

```
Dim childForm As Form
```

```
For Each childForm In Me.MdiChildren
    childForm.Text = "...."
```

```
Next
```

خاصية أخرى تعود بمرجع للنافذة الابن وهي الخاصية `ActiveMdiChild` والتي تمثل النافذة الابن النشطة:

```
Me.ActiveMdiChild.Text = "النافذة النشطة"
```

في المقابل، يمكنك تنشيط النافذة الابن باستدعاء الطريقة `ActivateMdiChild()` والتي تتطلب مرجع إلى النافذة الابن ترسله كوسيط لها:

```
Me.ActivateMdiChild(Me.MdiChildren(0))
```

المزيد أيضا، النماذج من النوع MDI يتم تفجير حدث إضافي لها هو `MdiChildActivate`، يتم تنفيذه بمجرد ان تحصل النافذة الابن على التركيز -أي تكون هي النافذة النشطة:

```
Private Sub MDIForm_MdiChildActivate(ByVal sender As Object, _
    ByVal e As System.EventArgs) Handles MyBase.MdiChildActivate
```

```
End Sub
```

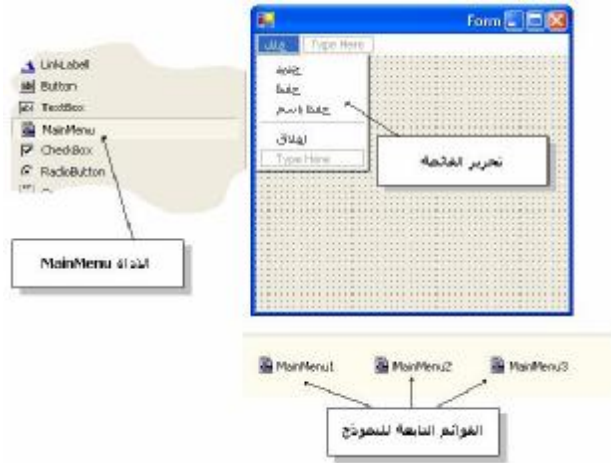
اخيرا، الطريقة `LayoutMdi()` تستخدم لترتيب النوافذ الابناء، حيث ترسل لها قيمة من اربع قيم للترتيب `MdiLayout` هي: ترتيب الرموز `ArrangeIcons`، تنالي `Cascade`، بجانب افقي `TileHorizontal`، وتجانب عمودي `TileVertical`:

```
Me.LayoutMdi(MdiLayout.ArrangeIcons)
Me.LayoutMdi(MdiLayout.Cascade)
Me.LayoutMdi(MdiLayout.TileHorizontal)
Me.LayoutMdi(MdiLayout.TileVertical)
```

القوائم Menus

في عالم Windows Forms، القوائم Menus ما هي إلا أدوات كالأدوات الموجودة في صندوق الأدوات (شكل 15-4)، يمكنك استخدام الأداة المعنونة `MainMenu` وإضافتها على نافذة النموذج، بل تستطيع إضافة أكثر من أداة `MainMenu` على نافذة النموذج و سيطرها لك مصمم

النماذج لرموز في أسفل نافذة النموذج، يمكنك تحديد الاداة ومن ثم البدء بتحريرها مرئيا Visually (شكل 13-9).



شكل 13-9: إضافة وتحرير القوائم.

يمكنك في أي وقت من حذف قائمة من القوائم التابعة للنموذج بالضغط بزر الفأرة الايمن على رمزها -أسفل النافذة- واختيار الامر Delete من القائمة المنبثقة، كما تستطيع تغيير خصائص وقت التصميم عن طريق نافذة الخصائص Properties Window، والتي يمكنك من تعديل خصائص اداة القائمة بشكل عام، أو خصائص كل عنصر من عناصر هذه القائمة.

ملاحظة

الرموز أسفل نافذة النموذج تسمى Designer's Tray Area تمثل نوعية من الأدوات لا تظهر بالشكل المتوقع لحظة التنفيذ. سترى الكثير منها في الفصل القادم **الأدوات Controls**.

ان احتوت نافذة النموذج على أكثر من أداة قائمة MainMenu، يمكنك اختيار احدها بإسناد قيمتها إلى الخاصية Menu والتابعة لنافذة النموذج، سواء وقت التصميم أو التنفيذ:

Me.Menu = MainMenu2

الخصائص، الطرق، والأحداث

فئات القوائم MainMenu مشتقة من الفئة Menu. من ناحية أخرى، عليك معرفة ان الفئة MainMenu تمثل اداة القائمة كلها بما تحتوي، اما بالنسبة لعناصر القائمة فهي كائنات من النوع MenuItem. وبمجرد إضافتك لعناصر جديدة، سيقوم مصمم النماذج بتوليد كائنات إضافية من النوع MenuItem لكل عنصر من هذه العناصر.

بالنسبة للخاصية Text فهي تمثل النص الظاهر على عنصر القائمة، يمكنك استخدام الحرف & وإتباعه بحرف خر يتم تسطيره Underline، حتى تمكن المستخدم من الضغط على المفتاح [Alt] وذلك الحرف لاختيار عنصر القائمة، يمكنك معرفة هذا الحرف وقت التنفيذ عن طريق الخاصية Mnemonic. كما تستطيع اسناد القيمة الحرفية "-" إلى الخاصية Text لعمل فاصل مجموعات للعناصر.

من الخصائص أيضا الخاصية Checked والتي تسند لها القيمة True لوضع علامة على عنصر القائمة، وهي شبيهة بالخاصية RadioCheck ولكنها تضع علامة دائرية وجرى العرف على ان تكون الوحيدة المعلمة في المجموعة.

يمكنك معرفة جميع العناصر الفرعية عن طريقة الخاصية MenuItemItems والتي تعود بمصفوفة من النوع MenuItem تمثل مراجع للعناصر الفرعية، كما تستطيع معرفة القائمة الحاضنة للقائمة الفرعية عن طريقة الخاصية Parent.

المزيد أيضا، يمكنك تحديد اختصار لوحة المفاتيح لتفعيل عنصر القائمة عن طريق الخاصية Shortcut، كما تستطيع اسناد القيمة False إلى الخاصية ShowShortcut ان اردت اخفاء نص اختصار لوحة المفاتيح المجانب لاسم العنصر.

بالنسبة للطرق، فهي تحتوي على مجموعة من الطرق أبرزها الطريقة GetMainMenu() التي تعود بكائن من النوع MainMenu يمثل اداة القائمة، وهناك الطريقة PerformClick() لتنفيذ الحدث Click التابع للعنصر، والطريقة PerformSelect() لتنفيذ الحدث Select التابع للعنصر.

وعلى ذكر الأحداث السابقة، فيوجد حدث النقر Click الذي يتم تنفيذه بمجرد اختيار القائمة من قبل المستخدم، و هناك أيضا الحدث Select الذي يتم تنفيذه ان كان المؤشر فوق العنصر (سواء كان مؤشر الفأرة أو لوحة المفاتيح).

القوائم المنبثقة Popup-Menu

يمكنك عمل قوائم منبثقة Popup-Menus بسهولة شديدة، تحتاج إلى استخدام الأداة ContextMenu عوضاً عن الأداة MainMenu، ويمكنك تحريرها بمصمم النماذج كما تفعل مع القوائم الرئيسية. أما إن أردت استخدامها، فعليك ربطها بالأداة وذلك بإسناد كائن النافذة المنبثقة إلى الخاصية ContextMenu التابعة للأداة (سواء وقت التصميم أو تنفيذ):

```
Button1.ContextMenu = ContextMenu1
```

تستطيع ربط نفس القائمة المنبثقة مع أكثر من أداة، وعند قيامك بربطها سيتم إظهارها إن قام المستخدم بالضغط بزر الفأرة الأيمن على الأداة. مع ذلك، لست بحاجة إلى انتظار نقرة المستخدم بزر الفأرة الأيمن لعرض القائمة المنبثقة، إذ يمكنك استدعاء الطريقة Show() وإرسال الأداة التابعة لها مع موقعها بالنسبة للأداة:

```
ContextMenu1.Show(Button1, New Point(0, 0))
```

انظر أيضاً

الفئة من النوع Point تمثل نقطة، لي عودة أخرى حول هذه الفئة في الفصل القادم **الأدوات Controls**.

تحتوي القوائم المنبثقة على الخاصية DefaultItem والتي تسند القيمة True لها إن أردت أن يكون عنصر القائمة هو العنصر الافتراضي Default Item، والعنصر الافتراضي ليس سوى عنصر مثل باقي العناصر ولا يميزه إلا طريقة كتابته بالزى السميك Bold فقط. أخيراً، عند ظهور القائمة المنبثقة ContextMenu، فسيتم تنفيذ حدث تابع لها وهو .Popup

نماذج MDI مرة أخرى

إذا كان لكلا النافذة الحاضنة MDI والنافذة المحضونة Child قائمة، فسيتم دمجها في قائمة واحدة تظهر في النافذة MDI، يمكنك التحكم في طريقة الدمج عن طريقة الخاصية MergeType -و المدعومة ليس فقط في الكائن MainMenu بل حتى عناصر القائمة MenuItem.

يمكنك اسناد قيمة من اربع قيم للخاصية MergeType هي: **Add**: سيتم اضافة القائمة إلى قائمة النافذة الحاضنة MDI كما هي، تحديد موقع العنصر يعتمد على ترتيبه في الخاصية MergeOrder. **Remove**: هذه القائمة لن تظهر ابدا في النافذة الحاضنة MDI. **Replace**: سيتم تبديل القائمة بالقائمة الموجودة في النافذة الحاضنة MDI والتي تحمل نفس الترتيب في الخاصية MergeOrder. **MergeItems**: سيتم دمج عناصر هذه القائمة مع عناصر قائمة في النافذة الحاضنة MDI والتي تحمل نفس الترتيب في الخاصية MergeOrder.

مثال للاستخدام الامثل:

عملية الدمج ليست بالصورة السهلة التي تتوقعها، حيث ان بها قليلا من التعقيد، وحتى ترى الاستخدام الامثل للخاصيتين MergeType و MergeOrder، فضلت عرض هذا المثال. بافتراض ان النافذة الحاضنة MDI تحتوي على هذه القائمة:

العنصر	الخاصية MergeType	الخاصية MergeOrder
ملف	MergeItems	0
جديد	MergeItems	0
فتح	Remove	1
حفظ	Remove	2
انهاء	MergeItems	9
أدوات	Add	5
خيارات	Add	0
تعليمات	MergeItems	9
المحتويات	MergeItems	0
حول...	MergeItems	2

وبافتراض ان النافذة الابنة Child تحتوي على هذه القائمة:

الخاصية MergeOrder	الخاصية MergeType	العنصر
0	MergeItems	ملف
0	Remove	جديد
1	Replace	فتح
2	Replace	حفظ
3	Add	اغلق
1	Add	تحرير
0	Add	نسخ
	9	MergeItems تعليمات
1	Add	الفهرس

عند الدمج، ستكون القائمة النهائية في النافذة الحاضنة MDI بهذا الشكل:

مدمجة من كلا النافذتين)	ملف
(من النافذة الحاضنة MDI)	جديد
(من النافذة المحظونة Child)	فتح
(من النافذة المحظونة Child)	حفظ
(من النافذة المحظونة Child)	اغلق
(من النافذة الحاضنة MDI)	انهاء
(من النافذة المحظونة Child)	تحرير
(من النافذة المحظونة Child)	نسخ
(من النافذة الحاضنة MDI)	أدوات
(من النافذة الحاضنة MDI)	خيارات
مدمجة من كلا النافذتين)	تعليمات
(من النافذة الحاضنة MDI)	المحتويات
(من النافذة المحظونة Child)	الفهرس
(من النافذة الحاضنة MDI)	حول...

الإنشاء الديناميكي للقوائم

عملية الإنشاء الديناميكي للقوائم لحظة التنفيذ لا اعتقد انها ستكون غريبة، فكل ما في الامر اننا نتعامل مع فئات تتطلب إنشاء كائنات منها، ومن ثم اسناد خصائص وقص أحداثها، وكل هذه المواضيع قد تطرقت لها في الفصل الثالث الفئات والكائنات من هذا الكتاب. الشيفرة التالية ستنشئ قائمة منبثقة ContextMenu وقت التصميم وقص جميع أحداث عناصرها باستخدام AddHandler ليتم تنفيذ الإجراء MenuClicked():

```
Private Sub Form1_Load(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles MyBase.Load

    Dim X As New ContextMenu()
    Dim sub1 As New MenuItem()
    Dim sub2 As New MenuItem()

    sub1.Text = "الامر 1"
    sub2.Text = "الامر 2"

    X.MenuItems.Add(sub1)
    X.MenuItems.Add(sub2)

    AddHandler sub1.Click, AddressOf MenuClicked
    AddHandler sub2.Click, AddressOf MenuClicked

    Button1.ContextMenu = X
End Sub

Sub MenuClicked(ByVal send As Object, ByVal e As EventArgs)
    MsgBox("تم اختيار الامر")
End Sub
```

مواضيع متقدمة

اختم معك الفصل الثالث عشر نماذج Windows Forms بهذا القسم، والذي أتطرق فيه إلى مواضيع متعددة هي : التفاعل مع نوافذ Modal، وراثته النماذج Form Inheritance، والنماذج المحلية Localized Forms.

التفاعل مع نوافذ Modeless

ذكرت في الفقرة طرق النموذج سابقا في هذا الفصل، انك تستدعي الطريقة Show() لفتح نافذة نموذج من النوع Modeless:

```
Dim MyForm As New Form2
```

```
MyForm.Show ( )
```

```
' الشيفرة التالية سيتم تنفيذها أيضا
```

```
...
```

```
...
```

وكما في التعليق السابق، سيستمر تنفيذ شيفرة الإجراء المستدعي بشكل متزامن مع إجراءات النافذة التي تم فتحها. في هذه الفقرة سنجيب على السؤال التالي: كيف يمكن للإجراء المستدعي من معرفة انه تم اغلاق النافذة التي فتحها؟

اجابة هذا السؤال نستنتجها من الفصل الثالث **الفئات والكائنات**، فكما أخبرتك مرارا ان نوافذ النماذج ما هي إلا فئات، وأحداثها هي أعضاء لتلك الفئات، لذلك يمكنك قنص الحدث Closing (أو أي حدث اخر) للتفاعل مع النوافذ من النوع Modeless:

```
Sub ShowForm ( )
```

```
Dim mdless As New Form1( )
```

```
AddHandler mdless.Closing, AddressOf FormHasBeenClosed
```

```
mdless.Show( )
```

```
End Sub
```

```
Sub FormHasBeenClosed(ByVal sender As Object, ByVal e As _  
System.ComponentModel.CancelEventArgs)
```

```
MsgBox(" تم اغلاق النافذة " & sender.text)
```

```
End Sub
```

وراثة النماذج Form Inheritance

تصميم النماذج بمصمم النماذج امر ممتع مبدئيا، ولكنه يصبح ممل جدا ان تكررت نفس التصميمات والشيفرات البرمجية الأولية، فمعظم صناديق الحوار تتكرر في أكثر من مشروع، مما يتطلب منك إعادة تصميمها وكتابة شفراتها من جديد.

بما ان نوافذ النماذج ما هي إلا فئات تقليدية، فيعني ذلك إمكانية اشتقاقها وراثيا وتطبيق الوراثة Inheritance عليها، وبذلك توفر على نفسك الكثير من الوقت المستخدم في التصميمات المكررة لنوافذ النماذج. ليس هذا فقط، بل حتى عند قيامك بتطوير نافذة النموذج القاعدية Base Form (اقصد فئة النموذج القاعدية)، سيشمل هذا التعديل كافة النماذج الوارثة منها، لذلك عند

اكتشافك لأحد الأخطاء في نافذة النموذج القاعدية، فلست بحاجة إلى تعديل كافة النماذج الوارثة منها.

لا يوجد شيء جديد اخبرك به عند وراثة النماذج، فكل ما تعلمناه في الفصل الرابع الوراثة يطبق مع النماذج بمرونة كبيرة. مع ذلك، ضع في عين الاعتبار ان فئات النماذج تشمل نوافذ تمثلها، وليس مجرد بيانات كالفئات الأخرى التقليدية.

نقاط إضافية:

من المهم معرفة ماذا تمثل الأدوات Controls التي تضعها بمصمم النماذج بالنسبة لفئة النموذج، اذ عليك ان تعلم علم اليقين ان كل اداة تضعها تمثل حقل Field في فئة النموذج، فلو وضعت اداة TextBox سيقوم مصمم النماذج بتعريفها كحقل:

```
Class Form1
    Inherits System.Windows.Forms.Form
    ...
    Friend WithEvents TextBox1 As Forms.TextBox
    ...
End Class
```

ينصح دائماً بتغيير محدد الوصول للحقول، وجعله Private حتى تحمي فئة النافذة من العبث بأدواتها من خارجها، أو Protected ان رغبت في السماح للنافذة المشتقة فقط من الوصول إلى اعضاء النافذة الحالية.

ملاحظة

بدلاً من تعديل محدد الوصول والخاص بالأداة يدوياً من خلال الشيفرة المصدرية، يمكنك استخدام خاصية الأداة Modifier من نافذة الخصائص.

استخدامك لمحدد الوصول Private سيحرم النافذة المشتقة من أشياء كثيرة منها: قنص أحداث الأدوات، إعادة تعريف طرقها وخصائص، والاهم من ذلك سيحرمها من الوصول إلى اعضاء الاداة ولن تستفيد منها في الحصول على أية معلومات.

مع ذلك، ان استخدمت محدد الوصول Private يمكنك السماح للنافذة المشتقة من الوصول إلى بعض عناصر الاداة عن طريق كتابتها يدويا بنفس:

```
Class Form1
    Inherits System.Windows.Forms.Form
    ...
    Private WithEvents TextBox1 As Forms.TextBox

    ' يمكن للفئة المشتقة الوصول إلى هذه الخاصية
    Friend Property TextValue() As String
        Get
            Return TextBox1.Text
        End Get

        Set(ByVal Value As String)
            TextBox1.Text = Value
        End Set
    End Property
    ...
End Class
```

مثال تطبيقي:

قم بتصميم نافذة نموذج، وضع الأدوات عليها واجري كافة التعديلات على خصائصها وتنسيقاتها، في (الشكل 13-10) صممت صندوق حوار بسيط خاص لكلمات المرور، اسم فنته PasswordForm.



شكل 13-10: تصميم مبدئي لنافذة قاعدية.

بعد تصميم صندوق الحوار، اختر الامر Build xxx (حيث xxx اسم المشروع) من قائمة Build، حيث لابد من ترجمة الملف حتى تتمكن من وراثة النافذة السابقة، أنشئ ملف جديد وقم فورا بوراثة النافذة PasswordForm بهذا الشكل:

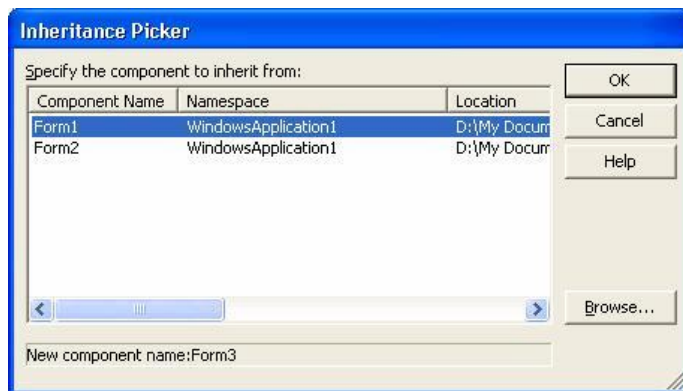
```

Class TestForm
    Inherits PasswordForm
    ...
    ...
End Class
    
```


ملاحظة

يمكنك وراثة النافذة أيضا دون الحاجة لاختيار الامر Build xxx لترجمة المشروع، ولكنك في هذه الحالة لن تتمكن من مشاهدة التأثيرات الأولية لوراثة النموذج في مصمم النماذج.

لست بحاجة لكتابة الشيفرة يدويا بنفسك، اذ يمكنك الاعتماد على بيئة التطوير Visual Studio .NET في وراثة النماذج، اختر الامر Add Inherited Form من القائمة Project، اكتب اسم الملف الذي تريده ثم اضغط على الزر Open، سيظهر لك صندوق حوار بعنوان Inheritance Picker (شكل 11-13).



شكل 11-13: صندوق الحوار Inheritance Picker.

حدد النافذة التي تود وراثتها واضغط على الزر OK، ستلاحظ ان نافذة نموذج جديدة ظهرت ولكنها ليست خالية، بل تحتوي على جميع أدوات وخصائص النافذة القاعدية (شكل 13-12)، وان أعمنت النظر ستري الرمز  في أعلى الزاوية اليسرى لكل اداة، ليعلمك ان هذه الاداة واثرة من النافذة القاعدية.



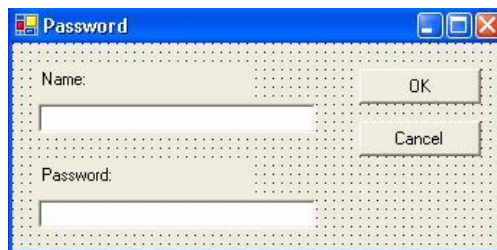
شكل 13-12: نافذة النموذج الجديدة قد ورثت النافذة القاعدية.

يمكنك تعديل خصائص الأدوات الذي سمح لك محدد الوصول المستخدم للأداة في النافذة القاعدية بذلك، فلو كان محدد وصول الأداة Private فلن تتمكن من عمل أي شيء، حتى لو حاولت فتح نافذة الخصائص لتغيير خصائص الاداة، ستلاحظ انها اصبحت باللون الخافت ولن تتمكن من تعديل أي خاصية فيها.

النماذج المحلية Localized Forms

حتى لو لم تكن تنوي تطوير تطبيقات متعددة اللغات، من الجيد إعطاء نكهة محلية لنماذج برنامجك، فقد يأتي يوم من الايام الذي تود تغيير واجهة نوافذ برنامجك ليغلب عليه الطابع المحلي للدولة المستخدمة، وبغض النظر عن مدى قبولك لهذه الفكرة، فالسهولة التي توفرها لك بيئة التطوير Visual Studio .NET لا تعطيك حجة لعدم فعل ذلك.

تخيل معي انني صممت نافذة لاستقبال كلمة المرور من المستخدم (شكل 13-13)، باللغة الإنجليزية، ووضعت في اعتباراتي انها لمستخدمين اجانب حيث محاذاة الأدوات وطريقة ترتيبها يغلب عليها طابع الاتجاه من اليسار إلى اليمين.

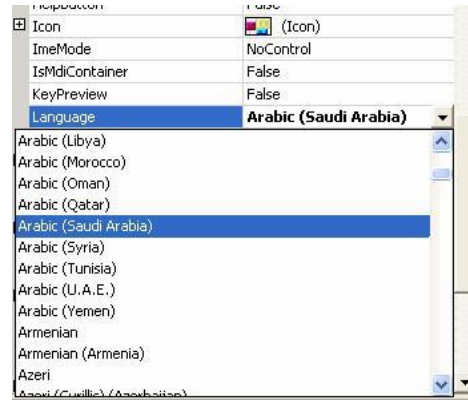


شكل 13-13: نافذة نموذج تقليدية.

هـب مثلاً اني بعد فترة اردت تغيير واجهة الاستخدام، وهذا الامر بحد ذاته سيكلف الكثير من الوقت والجهد، خاصة ان المسؤولية ستتخصر علي عند حصول أي خطأ، والسبب ان أي تعديل لمحتويات هذه النافذة ستتأثر فيه الشيفرة البرمجية، يبقى الحل الأسهل هو جعل هذه النافذة نافذة محلية Localized Form.

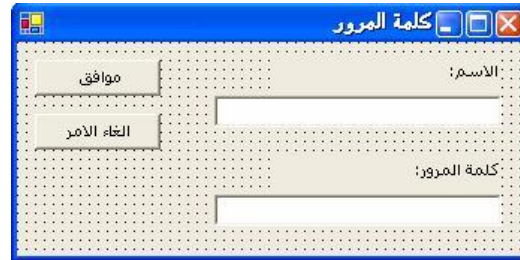
كل ما هو مطلوب منك لجعل النافذة محلية اسناد القيمة True إلى الخاصية Localizable من نافذة الخصائص (فهي ممكنة في وقت التصميم فقط)، وبمجرد إسنادك للقيمة True لهذه الخاصية فانك تطلب من مصمم النماذج تحويل جميع الشيفرات المولدة والتي تتعامل مع الخصائص إلى ملف المصادر Resource File، بحيث يمكن لبيئة التطوير من قراءة قيم الخصائص من هناك.

وحتى تفهم الهدف من الفكرة السابقة، حدد الدولة المراد دعمها في نافذة النموذج المحلية عن طريق الخاصية Language (شكل 13-14) من نافذة الخصائص (فهي أيضاً خاصة وقت التصميم فقط).



شكل 13-14: اختيار Arabic (Saudi Arabia) كأحد لغات النافذة.

بمجرد تحديثك للدولة في الخاصية Language قم بتصميم النافذة ليغلب عليها الطابع المحلي بالشكل الذي تريده، فالنافذة السابقة (شكل 13-13) قم بتعديل خصائصها وكتابتها باللغة العربية، كما أعدت ترتيب محاذاة الأدوات ليكون من اليمين إلى اليسار (شكل 13-15).



شكل 13-15: نافذة النموذج باللغة Arabic (Saudi Arabia)

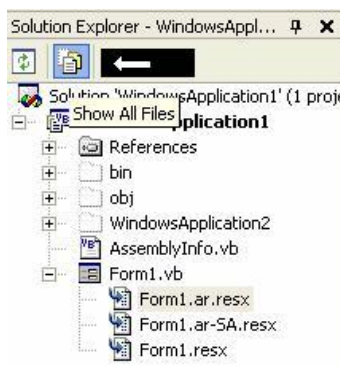
والان لدينا نافذة نموذج واحد ولكن بتصميمين مختلفين، يمكنك رؤية التصميم القديم (شكل 13-13) باختيار القيمة (Default) من الخاصية Language، وتستطيع العودة إلى التصميم الجديد (شكل 13-15) باختيار اللغة Arabic (Saudi Arabia) من نفس الخاصية. التعامل مع نوافذ النماذج المحلية هو نفس التعامل مع نوافذ النماذج التقليدية، ولا يوجد أي فرق جوهري يحتاج توضيحه، وان سألتني سؤال واخبرتني أي من النموذجين سيتم عرضه على المستخدم؟ فإجابتي ستكون بالاعتماد على الاعدادت الإقليمية في جهاز المستخدم، وان كانت

الاعدادات الإقليمية في جهاز المستخدم ليست مدعومة، فسيظهر النموذج (شكل 13-13) وذلك لان لغته هي الافتراضية (فقد صممته عندما كانت قيمة الخاصية Language هي (Default)).

نظرة خاطفة خلف الكواليس:

كما رأيت، عملية تدويل النماذج Internationalization Forms لا يتطلب إلا مجهود يسير جدا من طرف المبرمج، مع ذلك فهو يتطلب مجهود كبير من قبل بيئة التطوير Visual Studio .NET، حيث ان كل التصميمات مختلفة اللغات للنماذج، تحفظ قيم خصائصها في ملفات مصادر خاصة بنافذة النموذج، يحمل نفس اسم ملف النموذج بالإضافة إلى رمز الدولة مع الامتداد .resx في نفس المجلد.

يمكنك عرض ملف المصادر التابع لكل نافذة نموذج عن طريق نافذة مستكشف المشروع Solution Explorer، ولكن ذلك يتطلب منك الضغط على الزر العلوي لهذه النافذة والمسمى Show All Files (شكل 13-16).



شكل 13-16: عرضت ملفات المصادر بعد الضغط على الزر Show All Files.

لست هنا بصدد شرح ملفات المصادر، فيمكنك التجول فيها والبحث بمحتوياتها يدويا، ولكن دعني انوه هنا بان البيانات الموجودة في شكل جداول في ملفات المصدر، تحفظ في الخلفية بالصيغة XML (شكل 13-17 بالصفحة التالية). تستطيع تحرير الملفات بصيغة XML يدويا بالضغط على خانة التبويب XML في أسفل نافذة المصادر.

Data for data				
name	value	comment	type	parent
Label1.LineM	NoControl	(null)	System.Windows	(null)
Label1.LineM	174, 17	(null)	System.Windows	(null)
Label1.Text	الاسم	(null)	(null)	(null)
TextBox1.Lac	132, 36	(null)	System.Windows	(null)
TextBox1.Size	192, 21	(null)	System.Windows	(null)
TextBox2.Lac	132, 102	(null)	System.Windows	(null)
TextBox2.Size	192, 21	(null)	System.Windows	(null)
Label2.LineM	NoControl	(null)	System.Windows	(null)
Label2.Lac	132, 75	(null)	System.Windows	(null)
Label2.Text				
Button1	12, 6			
Button1	12, 7			
Button1	12, 8			
Button1	12, 9			
Button1	13, 0			
Button1	13, 1			
Button1	13, 2			
Button1	13, 3			
Button1	13, 4			
Button1	13, 5			
Button1	13, 6			
Button1	13, 7			

شكل 13-17: ملفات المصادر تحفظ بالصيغة XML.

ملاحظة

يمكنك مراجعة الشيفرة المولدة من قبل مصمم النماذج المحلية حتى ترى أمثلة وافية وشفافية لطريقة قراءة البيانات من ملفات المصادر باستخدام الفئة Resources.

نصحتي لك بأن لا تطوي غلاف الكتاب الآن، بل تابع واستمر في قراءة الفصل التالي الأدوات **Controls** فهو مرتبط ارتباطاً وثيقاً بهذا الفصل، وفيه عرض من الخصائص التي لا تعتبر خاصة بالأدوات وحسب، بل تنبع أيضاً للنماذج -حيث إن كلاهما مشتق من الفئة القاعدية **Control**.

الأدوات Controls

الاعتماد على نافذة النموذج وحدها لا يكفي لإنجاز تطبيقات Windows إنتاجية، حيث ان استخدام الأدوات Controls يعتبر جزء لا يتجزأ من مراحل تطوير برامجك المعتمدة على نماذج Windows Forms.

جميع الأدوات التي توضع في نماذج Windows Forms مشتقة وراثيا بشكل مباشر أو غير مباشر من الفئة Control (شكل 13-4 صفحة 449)، وبالتالي فجميع خصائص، طرق، وأحداث الفئة Control ستكون أيضا مدعومة في سائر الأدوات، لذلك وجدت انه من الأفضل - لي ولك - البدء بعرض الأعضاء المشتركة بين الأدوات (أعضاء الفئة Control) ومن ثم ذكر كل أداة على حده.

ملاحظة

لا تنسى ان هذه الخصائص، الطرق، والأحداث مدعومة أيضا في نافذة النموذج Form والتي تحدثت عنها في الفصل السابق، وذلك لان الفئة Form مشتقة وراثيا أيضا من الفئة Control.

الخصائص المشتركة

كما هو الحال مع نافذة النموذج، يمكن تعديل خصائص الأدوات وقت التصميم عن طريق نافذة الخصائص Windows Properties (شكل 14-5) وذلك بعد تحديد الأداة، ليقوم مصمم النماذج بتوليد الشيفرة الضرورية في الإجراء InitializeComponent() والخاص بصمم النماذج. الشيفرة التي ولدها مصمم النماذج لحظة تعديل الخصائص، علمتي أشياء كثيرة وسرعت علي استكشاف عشرات الكائنات وطرق استخدامها لحظة كتابة هذه السطور التي تقرأها الآن.

لذلك، لا تفوت الفرصة على نفسك في استكشاف الشيفرات التي يولدها مصمم النماذج عندما تغير كل خاصية من خصائص الأدوات.

ملاحظة

لا اقصد بكلمة مشتركة (الخصائص المشتركة، الطرق المشتركة، والأحداث المشتركة) بالأعضاء المشتركة Shared Members. وإنما اقصد أعضاء مشتركة بين الأدوات Common Members فهي مشتقة من الفئة Control.

اسم الأداة Name

قد تجد الخاصية Name في أعلى نافذة الخصائص، مع ذلك من الخطأ اعتبارها خاصية، فالقيمة Name التي نضعها في هذه الخانة تمثل الاسم البرمجي للأداة، وعند تغيير الاسم سيقوم مصمم النماذج بتعديل كافة الشيفرات التي استخدمت اسم الأداة في الإجراء InitializeComponent(). نصيحة أخوية، لا تحاول تعديل اسم الأداة بعد كتابة الكثير من الشيفرات المصدريّة، وذلك سيضطرك إلى تعديل جميع الشيفرات التي استخدمت الأداة بنفسك، حيث -كما قلت- ان مصمم النماذج يعدل الاسم البرمجي للأداة تلقائياً في الإجراء InitializeComponent() فقط. نصيحة أخوية أخرى، لا تعتمد على الأسماء الابتدائية Text1، Text2، Text3 فهي ستسبب لك التشويش وإضعاف قدرة تذكرها، خاصة ان كثرت الأدوات على سطح النافذة. لست بحاجة إلى تذكيرك ان شروط تسمية الأدوات يطبق عليها شروط تسمية باقي المعارف الأخرى (فهي أسماء برمجية لكائنات)، ويمكنك العودة إلى الفصل الثاني لغة البرمجة لقراءة شروط التسمية.

خصائص المظهر

نبدأ بالخاصتين Visible و Enabled كلاهما يحمل قيمة منطقية Boolean تمثل الاول ظهور أو عدم ظهور الأداة، والثانية تمكين أو عدم تمكين الأداة، ان أسندت القيمة False للخاصية Enabled فستظهر الأداة بلون خافت يوحي بان المستخدم لن يستطيع استخدامها وبالتسالي فلن تطلق الأداة اي أحداث وقت التنفيذ، مع ذلك لديك فرصة كبيرة في الوصول إلى الأداة برمجياً باستدعاء طرقها واسناد قيم لخصائصها.

الخاصية Text حروفية تمثل النص الظاهري على جبهة الأداة، نوع وحجم خط هذا النص يعتمد على الخاصية Font والتي سأحدث عنها بعد قليل. بالنسبة للخاصية RightToLeft فتستطيع إسناد القيمة RightToLeft.Yes لها ان كنت تنوي إسناد حروف عربية على الأداة لتظهر في السياق اليمين إلى اليسار Right-To-Left. كما يفضل إسناد القيمة RightToLeft.Inherit لها إن أردت تغيير اتجاه السياق بشكل تلقائي بحيث يتوافق مع سياق نافذة النموذج أو الأداة الحاضنة للأداة. اما الخاصية TextAlign ففيها تحدد محاذاة النص، والذي يكون قيمة من تسع قيم (شكل 14-1).

TopLeft	TopCenter	TopRight
MiddleLeft	MiddleCenter	MiddleRight
BottomLeft	BottomCenter	BottomRight

شكل 14-1: القيم التسعة الممكنة للخاصية TextAlign.

ملاحظة

ان كان سياق الأداة في الخاصية RightToLeft من اليمين إلى اليسار، فسيتم عكس مواقع القيم xxxLeft و xxxRight للخاصية TextAlign. اي ستكون القيمة TopRight تمثل الزاوية العليا اليسرى وليس اليمنى.

القيم التسع السابقة تابعة للتركيب ContentAlignment والذي يطبق على الأدوات Label، Button، CheckBox، و RadioButton فقط. اما الأدوات الأخرى فستعامل مع ثلاث قيم هي: Left، Right، و Center تابعة للتركيب HorizontalAlignment. اما خاصية الخط Font فهي تحمل قيمة تمثل كائن من النوع System.Drawing.Font. يحتوي على مجموعة كبيرة من الخصائص والطرق والتي سأفصلها في الفصل القادم GDI+، يمكنك تعديل الخط عن طريق صندوق الحوار Font والذي يظهر زر على شكل ... في نافذة الخصائص، أو يمكنك انشاء كائن من الفئة System.Drawing.Font:

```
TextBox1.Font = New Font("Tahoma", 20, FontStyle.Bold Or _
    FontStyle.Italic)
```

مشكلة بسيطة حلها ايسر، وهي ان الكائن المسند إلى الخاصية Font لا يمكنك تعديل قيمه، فمعظم خصائص الفئة Font للقراءة فقط ReadOnly، لذلك عليك إسناد قيمة كائن جديدة باستخدام المعامل New:

```
' زيادة حجم الخط إلى الضعف '
With TextBox1
    .Font = New Font(Font.Name, .Font.Size * 2, .Font.Style)
End With
```

ملاحظة

مشيد الفئة Font تم إعادة تعريفه Overloads بثلاث عشرة صيغة مختلفة. الصيغة التي استخدمتها في الشيفرتين السابقتين تستقبل 3 وسيطات الاولى اسم الخط، الثانية حجمه، والثالثة نمطه.

نقطة هامة علي توضيحها هي ان خاصية الخط Font لجميع الأدوات تشير إلى نفس كائن الخط التابع لنافذة النموذج أو الأداة الحاضنة ما لم تغير الخاصية Font لكل أداة محضونة، ماذا نستنتج من هذا الكلام؟ نستنتج ان اي تغيير لخاصية الخط التابعة لنافذة النموذج أو الأداة الحاضنة سيتم تغيير جميع الخطوط التابعة للأدوات المحضونة بها بشكل تلقائي. لذلك، إن أردت أن تكون الأداة محضونة مستقلة بخاصيتها Font (بحيث لا تتأثر ان تغيرت الأداة الحاضنة) وان تحمل نفس قيم الأداة الحاضنة بشكل ابتدائي، يمكنك نسخ الكائن باستخدام Clone():

```
TextBox1.Font = Form1.Font.Clone()
```

خصائص الموقع والحجم

قبل البدء في سرد خصائص الموقع والحجم، من الجيد تعريفك بمجموعة من الفئات الاولى System.Drawing.Point والتي تمثل نقطة تحتوي على خاصيتين للقراءة فقط ReadOnly هما الإحداثي السيني X (والذي يزيد كلما اتجهنا إلى اليمين) والإحداثي الصادي Y (الذي يزيد كلما اتجهنا إلى الأسفل)، والفئة الثانية System.Drawing.Size التي تحتوي على الخاصيتين للقراءة فقط أيضا هما Width و Height تمثلان عرض وارتفاع الأداة. اما الفئة الثالثة فهي

System.Drawing.Rectangle والتي تمثل منطقة Client Region تحتوي على مجموعة من الخصائص والطرق أيضا تشمل موقع، حجم، مساحة وغيرها من المعلومات والخاصة بالمنطقة.

ملاحظة

الوحدة المستخدمة بشكل افتراضي في القياسات هي البكسل Pixel والذي يمثل نقطة رسم واحد من الشاشة.

اسند قيمة من النوع Point إلى الخاصية Location لتحدد موقع الأداة بالنسبة لنافذة النموذج أو الأداة الحاضنة لها، واسند قيمة من النوع Size إلى الخاصية Size للتحكم في حجم الأداة، يمكن تعديل هذه الخصائص باستخدام الفأرة في مصمم النماذج أو كتابة الشيفرة التالية (والتي يولدها المصمم بشكل تلقائي):

```
نقل الأداة إلى الزاوية العليا اليسرى '
TextBox1.Location = New Point(0, 0)

عرض الأداة 400 وارتفاعها 100 '
TextBox1.Size = New Size(400, 100)
```

ملاحظة

توجد خصائص قديمة هي Left، Top، Height، و Width صممت خصيصا للتوافقية مع الإصدارات (6->1) Visual Basic بل وأضيفت الخصائص Bottom و Right للتحكم في موقع وحجم الأداة. مع ذلك، لا أنصحك باستخدامها فهي أسلوب قديم ولن يقدم لك اي نتائج ايجابية عند الحديث عن تحسين الكفاءة Optimization.

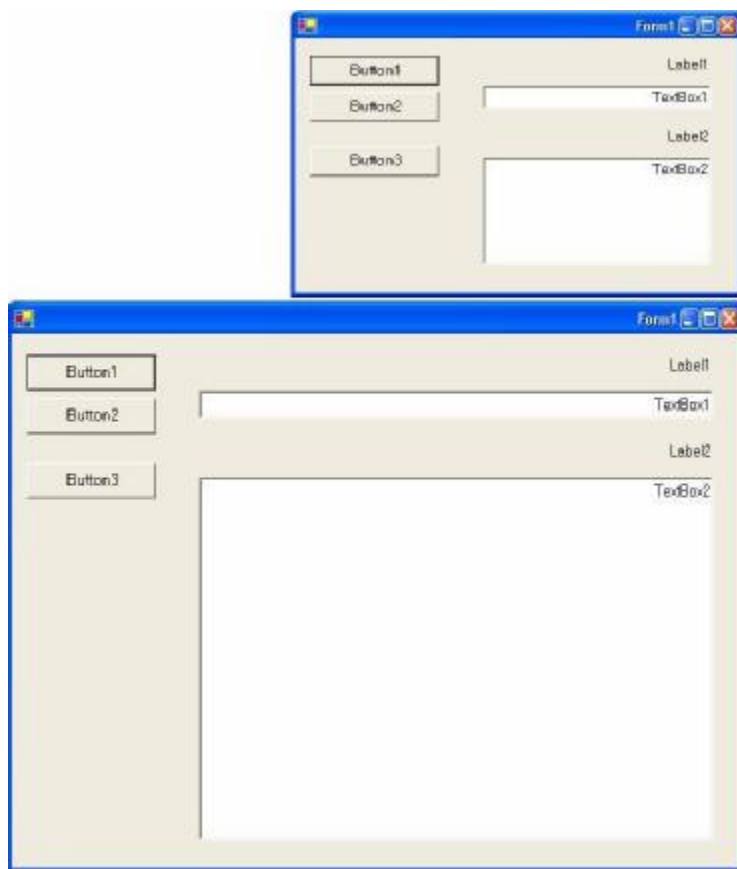
من خصائص الموقع والحجم خصائص من النوع System.Drawing.Rectangle كالخاصية Bounds والتي تمثل المنطقة الكلية للأداة (شاملة حدودها وموقعها)، والخاصية ClientRectangle التي تمثل المنطقة الداخلية للأداة (متجاهلة حدودها وموقعها). الشيفرة التالية ستجعل الأداة تغطي كافة المنطقة الداخلية للنافذة:

```
TextBox1.Bounds = Form1.ClientRectangle
```

الخاصية Anchor:

بالنسبة للخاصية Anchor فهي خاصية مفيدة توفر لك كتابة عشرات الاسطر البرمجية لتنسيق موقع الأداة ان تغير حجم النافذة، حيث تثبت المسافة بين حدود الأداة وبين الحد الخارجي لنافذة النموذج أو الأداة الحاضنة في حالة ما تغير حجمها، يمكنك تثبيت المسافة من الأعلى، الأسفل، اليمين، أو اليسار بإسناد القيم AnchorStyles.Top، AnchorStyles.Bottom، AnchorStyles.Right، أو AnchorStyles.Left. كما تستطيع الغاء تثبيت المسافة بإسناد القيمة AnchorStyles.None.

ان قمت بتثبيت المسافة لجهتين متضادتين (كفوق وتحت، أو يمين ويسار) فذلك سيضطر الأداة إلى تغيير حجمها حتى تثبت المسافة بين حدودها وبين الحد نافذة النموذج أو الأداة الحاضنة. وهذه فكرة رائعة جدا جدا لتمكين الأداة تحجيم نفسها بشكل تلقائي ان قام المستخدم بتغيير حجم النافذة (كما تفعل أداة الشجرة Tree في يسار مستكشف النظام Windows Explorer). يمكنك الاستفادة من نتيجة التضادات السابقة بإنشاء صناديق حوار قابلة للتمدد والنقل كما ترى في (الشكل 14-2 بالصفحة المقابلة):

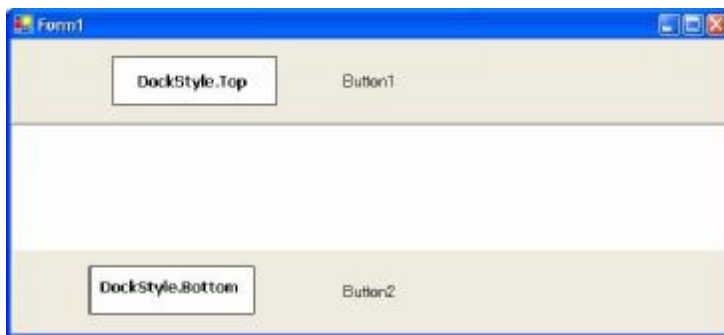


شكل 14-2: تأثير الخاصية Anchor.

حتى تعرف كيف استطعت انجاز صندوق الحوار السابق، فدعني اخبرك بان القيم كانت للأدوات كالتالي: TextBox1 هي: Top، Right، و Left، الأداة TextBox2 هي: Top، Bottom، Right، و Left، أدوات Label هي: Top و Right، اما الأدوات Button فكانت قيم خاصيتها Anchor هي: Top و Left.

الخاصية Dock:

اما الخاصية Dock فهي تمكنك من تغيير حجم وموقع الأداة بالنسبة لنافذة النموذج أو الأداة الحاضنة بحيث تحاذيها اما من اعلى DockStyle.Top كما تفعل اشربة الأدوات ToolBars، من الاسف DockStyle.Bottom كما هو الحال مع سطر الحالة StatusBar (شكل 14-3)، من اليمين DockStyle.Right، من اليسار DockStyle.Left، أو تغطي كامل النافذة أو الأداة الحاضنة DockStyle.Fill.



شكل 14-3: تأثير الخاصية Dock.

تستطيع التحكم في المسافة بين الأداة التي تم محاذاتها والأداة الحاضنة لها عن طريق الخاصية DockPadding للأداة الحاضنة أو نافذة النموذج، يمكنك تحديد المسافة من الحد الايسر للأداة الحاضنة بهذه الطريقة:

```
Form1.DockPadding.Top = 10
```

وكافة الجهات الاخرى Left، Right، و Bottom بنفس الطريقة مع انك تستطيع إسناد قيم الجهات الاربع كلها بضربة واحدة:

```
' تعادل
' Form1.DockPadding.Top = 10
' Form1.DockPadding.Bottom = 10
' Form1.DockPadding.Left = 10
' Form1.DockPadding.Right = 10
```

```
Form1.DockPadding.All = 10
```

ملاحظة

الخاصية DockPadding مدعومة في الأدوات الحاضنة فقط، وهي الأدوات المشتقة من الفئة ContainerControl (شكل 13-4 صفحة 449).

خصائص الاحتضان

طريقة وخاصية الاحتضان FindForm() و Parent تعودان بمرجع إلى نافذة النموذج الحاضنة للأداة في الطريقة FindForm() (احذر فـ FindForm() طريقة Method وليست خاصية)، أو الأداة الحاضنة للأداة في الخاصية Parent. بالنسبة للخاصية Parent، فيمكن إسناد أي قيمة لها لتغيير الأداة الحاضنة للأداة الحالية (حتى لو اردت تغيير نافذة النموذج الحاضنة لها!):

```
Dim frmForm2 As New Form2

TextBox1.Parent = frmForm2
frmForm2.Show ()
```

مع ذلك، حتى وان نقلت الأداة إلى نافذة أخرى فلا تنسى ان الاسم البرمجي للأداة لا يزال يتبع إلى فئة نافذة النموذج الاصلية Form1.TextBox1 (حيث ان الأدوات قد عرفت في فئة النموذج لقص أحداثها باستخدام WithEvents كما سترى لاحقاً)، الا انك تستطيع إسناد مؤشر كائن اخر في النافذة الاخرى وتتبع الأداة).

لا تنسى ان الطريقة FindForm() والخاصية Parent تمثل مرجع إلى نافذة النموذج الحاضنة أو الأداة الحاضنة لذلك يمكنك الوصول إلى طرقها وخصائصها بدون أي إشكالية:

```
TextBox1.FindForm().Test = "النافذة الحاضنة"
```

نافذة النموذج والأدوات الحاضنة الأخرى Panel و GroupBox مشتقة وراثياً من الفئة ContainerControl (شكل 13-4 صفحة 449)، لذلك تحتويان على خصائص إضافية تتعلق بالاحتضان، من هذه الخصائص الخاصة Controls والتي تمثل مرجع إلى جميع الأدوات المحضونة في الأداة الحالية.

ضع في عين الاعتبار ان الخاصية Controls عبارة عن مجموعة Collection تمثل الأدوات المحضونة في أعلى مستوى، والذي اقصد من العبارة اعلى مستوى أي أدوات

المحفوظة بها فقط ولا تشمل الأدوات المحفوظة بتلك الأدوات المحفوظة في الأداة نفسها، وحتى أسهل لك استيعاب الفكرة افترض هذه الأدوات الحاضرة والمحفوظة:

```
Form1
  GroupBox1
    TextBox1
    TextBox2
    TextBox3
  GroupBox2
    TextBox4
    TextBox5
```

الخاصية Form1.Controls ستعود بمرجع يمثل الأداةين GroupBox1 و GroupBox2 فقط، والخاصية GroupBox1.Controls ستعود بمرجع يمثل الأدوات TextBox1، TextBox2، و TextBox3، بينما الأداةين TextBox4 و TextBox5 هي المراجع التي ستعود بها الخاصية GroupBox2.Controls.

خصائص الألوان

الخاصيتان ForeColor و BackColor تمثلان لون الأمامية والخلفية للأداة، لون الامامية - بالتحديد- هو لون النص الظاهر على الأداة، ولون الخلفية هو لون سطح الأداة. الألوان في عالم Windows بشكل عام و NET. بشكل خاص تصنف إلى قسمين رئيسين هما: **الوان النظام System Colors**، و **الوان الخاصة Custom Colors**.

الوان النظام هي الالوان المفضلة في معظم الاحوال، فهي الوان حددها المستخدم في لوحة التحكم Control Panel لتظهر به سائر نوافذ وأدوات تطبيقات Windows بها. لذلك فمن باب احترام وتقدير ذوق المستخدم استخدام نفس الالوان التي طلبها. تجد هذه الالوان في الفئة System.Drawing.SystemColors:

```
TextBox1.BackColor = SystemColors.Window
TextBox2.BackColor = SystemColors.ActiveBorder
```

اما الالوان الخاصة فهي ستاتيكية لا تتغير مهما كانت اعدادات نظام التشغيل، وان لم تكن مصمم راقي فأتمنى من صميم قلبي ان تتجاهل هذه الفقرة، اما ان كنت ممن اتخذ في فن الرسم سبيلا، فيمكنك الاعتماد على الفئة System.Drawing.Color وتحديد الالوان باسمها:

```
TextBox1.BackColor = Color.Black
TextBox1.ForeColor = Color.White
```

أو تحديد العمق اللوني للأحمر، الأخضر، والأزرق باستدعاء الطريقة FromArgb() والتابعة لنفس الفئة السابقة:

```
TextBox1.BackColor = Color.FromArgb (255, 0, 0)
```

ملاحظة

الخاصيتان BackColor و ForeColor تشيران إلى نفس خصائص نافذة النموذج أو الأداة الحاضرة (حالتها كحال الخاصية Font). لذلك، أي تأثير على خصائص الأدوات الحاضرة سيشمل الأدوات المحضونة ما لم تسند قيم خاصة للأدوات المحضونة وقت التصميم.

خصائص التركيز

اقصد بكلمة التركيز Focus هي قدرة الأداة على ان تكون الأداة النشطة Active Control ويكون التركيز Focus عليها (كأداة TextBox عندما تبدأ الكتابة بها). يمكنك معرفة ما اذا كانت الأداة قابلة لاستقبال التركيز عليها عن طريق الخاصية CanFocus القابلة للقراءة فقط ReadOnly. ويمكنك معرفة ما اذا كان الأداة هي الأداة النشطة وعليها التركيز فعلا عن طريق الخاصية Focused وهي للقراءة فقط أيضا. الأداة التي عليها التركيز تسمى الأداة النشطة، يمكن للأداة (والحديث هنا عن الأداة الحاضرة أو نافذة النموذج) من معرفة الأداة النشطة والمحضونة فيها عن طريق الخاصية ActiveControl والتي تعود بمرجع يمثل الأداة النشطة (الخاصية قابلة للكتابة أيضا).

خصائص الجدولة

معظم مستخدمي تطبيقات Windows يفضلون استخدام مفتاح الجدولة [TAB] لنقل التركيز بين الأدوات، معظم الأدوات التي لها قابلية انتقال التركيز Focus عليها تحتوي على الخاصيتين TabStop و TabIndex. أسند القيمة True إلى الخاصية TabStop إن أردت من الأداة ان تستقبل التركيز بمجرد ان يضغط المستخدم على المفتاح [TAB]، ثم حدد ترتيب وتسلسل الأدوات في الخاصية TabIndex مع العلم ان الترقيم يبدأ بصفر.

ملاحظة

حتى لو أسندت القيمة False إلى الخاصية TabStop، فإن للمستخدم فرصة كبيرة لنقل التركيز على الأداة بالنقر عليها بزر الفأرة.

خصائص أخرى

من الخصائص الأخرى التي أود ذكر أسمائها فقط الخصائص Disposing، Created، و Disposed والتي تعود بالقيمة True في حال ما -على التوالي- تم انشاء الأداة فعلا، الأداة على وشك الموت، الأداة ماتت فعلا.

يمكنك إسناد قيمة تمثل قائمة Menu إلى الخاصية ContextMenu يتم عرضها على شكل قائمة منبثقة Pop-up menu.

هناك خصائص أخرى تفيدك في حالة قيامك بتطوير أدوات خاصة Custom Controls كالخاصيتين ProductName، ProduceVersion، و CompanyName.

لديك خاصية الاقفال Locked (والتي تستخدم لحظة التصميم فقط من قبل مصمم النماذج) حيث تثبت الأداة وتمنعك من تحريكها أو تغيير حجمها بطريق الخطأ.

أخيرا، مجموعة من الخصائص موجه لذوي الاحتياجات الخاصة (عافاني الله وإياك) هي AccessibleName، AccessibleDescription، AccessibleRole، و IsAccessible - راجع مكتبة MSDN لكيفية الاستفادة منها عسى أن لا تحتاجها يوما من الأيام.

الطرق المشتركة

معظم الطرق ليست سوى نسخ مكررة من الخصائص، فمثلا الطريقة SetSize() ترسل مع وسيطاتها عرض وارتفاع الأداة، كذلك الطريقة SetBounds() والتي تشمل في وسيطاتها الأحدثي السيني والصادي للأداة وعرضها وارتفاعها بضربة واحدة.

لديك الطريقتان BringToFront() و SendToBack() التي تظهر الأداة فوق الأدوات الأخرى أو خلف الأدوات الأخرى، مع العلم أن الأدوات المحضونة تكون دائما فوق الأداة الحاضنة.

الطريقة Show() تظهر الأداة (تعادل القيمة True للخاصية Visible)، والطريقة Hide() تخفي الأداة (تعادل القيمة False للخاصية Visible).

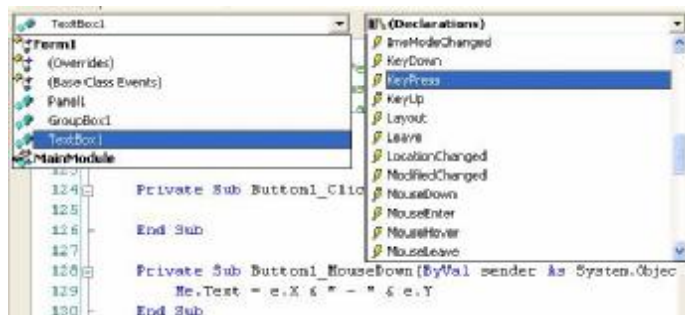
لو تذكر ان بعض خصائص الأدوات تتأثر بقيمة خصائص الأداة الحاضنة لها (كالخصائص ForeColor و BackColor) ما لم تتغير قيمة الخاصية للأداة، يمكنك استرجاع قيم الخصائص لتشير إلى كائنات الأدوات الحاضنة إما بإسناد قيم الكائنات بنفسك، أو استدعاء الطرق ResetForeColor() و ResetBackGround().

يمكنك توجيه التركيز إلى الأداة (القابلة لاستقبال التركيز) باستدعاء الطريقة Focus()، كما تستطيع معرفة الأداة التالية والتي سيأتي دورها في الخاصية TabIndex لمفتاح الجدولة [TAB] باستدعاء الطريقة GetNextControl() والتي تعود بمرجع للأداة التالية.

أخيراً، الطرق Invalidate()، Refresh()، و Update() تقوم بإعادة رسم الأداة، بإمكانك مراجع مكتبة MSDN لتوضيح الفروق بينها.

الأحداث المشتركة

في هذا القسم نعرض مجموعة كبيرة من الأحداث بشكل سريع. وكما هو الحال مع النماذج، يمكنك الاستعانة بمحرر الشيفرة لقنص الأحداث دون الحاجة لكتابتها بنفسك. ولكن هذه المرة باختيار اسم كائن الأداة من القائمة العلوية اليمنى، ومن ثم الحدث المراد قنصه من القائمة العلوية اليسرى (شكل 14-4).



شكل 14-4: الاستعانة بمحرر الشيفرة لقنص أحداث الأدوات.

أحداث الفأرة

معظم شيفراتك المصدريّة يتم تنفيذ نتيجة لأعمال درامية قام بها مستخدم البرنامج بالفأرة، من هذه الأحداث الحدث Click والذي يتم تنفيذه لحظة النقر على الأداة والحدث DbClick لحظة النقر المزدوج.

يمكنك معرفة المزيد من التفاصيل حول الفأرة بالاستعانة بالحدث MouseDown والذي يتم تنفيذه بمجرد الضغط بزر الفأرة على الأداة (النقر هي عملية الضغط بزر الفأرة ومن ثم تحريره، بينما الضغط لا يشترط تحرير الزر لتنفيذ الحدث)، من هذه التفاصيل الأزرار التي تم ضغطها بالفأرة في الخاصية Button لوسيلة الحدث:

```
Private Sub Button1_MouseDown(ByVal sender As Object, _
    ByVal e As Forms.MouseEventArgs) Handles Button1.MouseDown

    If e.Button = MouseButton.Left Then
        'الزر الايسر
        ...
    ElseIf e.Button = MouseButton.Middle Then
        'الزر الاوسط
        ...
    ElseIf e.Button = MouseButton.Right Then
        'الزر الايمن
        ...
    End If
End Sub
```

كما تستطيع معرفة احداثيات موقع الفأرة إما من الأداة (باستخدام الخصائص X و Y التابعة لوسيلة الحدث) أو بالنسبة للنافذة (باستخدام الخاصية المشتركة Control.MousePosition):

```
Private Sub Button1_MouseDown(ByVal sender As Object, _
    ByVal e As Forms.MouseEventArgs) Handles Button1.MouseDown

    Me.Text = e.X & " - " & e.Y
End Sub
```

من التفاصيل التي تحصل عليها من وسيلة الحدث MouseDown، عدد مرات النقر منذ آخر تنفيذ للحدث في الخاصية Click، والخاصية Delta التي تمكنك من معرفة مقدار تحرك عجلة الفأرة Mouse Wheel.

وسيطرة الحدث `MouseDown` هي من النوع `MouseEventArgs`، وهي نفس الوسيلة التي يتم إرسالها إلى الأحداث `MouseUp`، `MouseMove`، و `MouseWheel`. يتم تنفيذ الحدث الأول في حالة تحرير زر الفأرة، الثاني عن تحريك الفأرة فوق الأداة، والثالث عن تحريك عجلة الفأرة على الاداة في حل ما إذا كان التركيز عليها. العدد الكلي لأحداث الفأرة هو تسع أحداث وقد ذكرنا ستا منها، اما الثلاثة الباقية `MouseEnter`، `MouseLeave`، `MouseHover` فيتم تنفيذها مع بداية دخول الفأرة على الأداة، خروج الفأرة عن حدود الأداة، أو عند تحريك الفأرة على الأداة (كما هو الحال مع الحدث `MouseMove`).

أحداث لوحة المفاتيح

ثلاثة أحداث مرنة توفرها لك الأدوات تتعلق بلوحة المفاتيح هي: `KeyUp`، `KeyDown`، و `KeyPress`. بالنسبة للحدث الثالث فيتم تنفيذه لحظة النقر على مفتاح من مفاتيح لوحة المفاتيح، ترسل وسيطة هذا الحدث خاطبتين هما `KeyChar` و `Handled`، الخاصية الأولى تمثل الحرف الذي تم النقر عليه، ويمكنك إسناد القيمة `True` إلى الخاصية الثانية إن أردت إخبار إطار عمل `NET`. انك قمت بعمل اللازم لردة الفعل على هذا المفتاح ولا تريد منه أي يقوم بأي عمل إضافي، بمعنى اخر اسنادك للقيمة `True` إلى هذه الخاصية يلغي عملية النقر وكأن شيئاً لم يحدث. (تفيدك بكثرة مع الأداة `TextBox` إن أردت الغاء مدخلات المستخدم الغير مناسبة).

بالنسبة للحدثان `KeyUp` و `KeyDown` فيتم تنفيذهما بمجرد قيام المستخدم بالضغط على الزر و تحريره، ضع في عين الاعتبار ان الحدث `KeyDown` سيتم تنفيذه بشكل متكرر طالما كان الزر مضغوطاً.

يرسل كلا الحدثين وسيطة من النوع `KeyEventArgs` تحتوي على الخصائص `Alt`، `Shift`، و `Control` لتعود بقيم تبين حالة ضغط المفاتيح `[Alt]`، `[Shift]`، و `[Ctrl]` على التوالي. كما تحتوي على الخاصية `KeyCode` التي تمثل المفتاح بقيمته الفيزيائية الذي تم ضغطه (نوع القيمة هذه تركيب `Enum` بالاسم `Keys`)، كما تحتوي الوسيلة على الخاصية `Handled` والتي مثل الخاصية `Handled` التابعة لوسيلة الحدث `KeyPress`.

فرق اخر بين الحدث `KeyPress` والحدثين `KeyUp` و `KeyDown` وهو ان الحدث `KeyPress` يتم تنفيذه ان ضغط المستخدم على الحروف أو الارقام المطبوعة، المفتاح `[Enter]`، المفتاح الجدولة `[Tab]`، والمفتاح `[Esc]` فقط، اما باقي المفاتيح كمفاتيح الوظائف `[F1]`، `[F2]`،

...الخ مفاتيح الاسهم، المفاتيح [Shift]، [Ctrl]، [Alt]... الخ فالحدث KeyDown لها بالمرصاد.

يوجد حدث اضافي يماثل الحدث KeyPress السابق وهو HelpRequested والذي يتم تنفيذه في حال ما قام المستخدم بالنقر على المفتاح [F1].
اخيرا، لو تم كتابة شيفرات في أحداث لوحة المفاتيح لكلا الأداة ونافذة النموذج، و اردت معرفة ايهما سيبدأ في ردة الفعل، فستكون نافذة النموذج هي الأولى ان كانت خاصيتها KeyPreview تساوي True، بينما تستجاب أحداث الأداة اولا ان كانت قيمة الخاصية False.

أحداث التركيز

عندما تستقبل الأداة التركيز، فسيتم تنفيذ حدثها الخاص وهو GotFocus، وعلى العكس ان فقدت الأداة التركيز سيتم تنفيذ حدثها LostFocus.

تفضل مستندات Microsoft .NET Documentation بالاعتماد على الحدثين Enter و Leave، يتم تنفيذ الأول بمجرد وقوع التركيز على الأداة (قبل الحدث GotFocus)، والثاني بعد فقد الأداة تركيزها (قبل الحدث LostFocus).

السبب الذي يجعلك تفضل استخدام الحدثين Enter و Leave عن GotFocus و LostFocus تجنب التشويش المنطقي في ترتيب وقوعها في سلسلة أحداث التركيز، حيث ترتيب السلسلة يكون كالتالي:

Enter <- GotFocus <- Leave <- Validating <- Validated.

بالنسبة للحدثين Validating و Validated فيفيدانك كثيرا عند التعامل مع الأداة TextBox، لذلك فضلت تأجيل مثال لاستخدامهما عند فقرة الأداة TextBox لاحقا في هذا الفصل.

أحداث أخرى

من الأحداث أيضا، حدث الرسم Paint - الذي ذكرته في الفصل السابق نماذج Windows Forms - والذي يتم تفجيرها كلما دعت الحاجة إلى إعادة رسم الأداة، والحدث Resize الذي يتم تنفيذه بمجرد تغيير حجم الأداة، والحدث Move الذي يتم تنفيذه ان تم تحريك الأداة (يفيدك الحدث الاخير كثيرا مع نوافذ النماذج).

يوجد حدثان اضافيان يتعلقان بالأدوات الحاضرة هما ControlAdded و ControlRemoved حيث يتم تنفيذ الاول في كل مرة تضيف أداة محضونة إلى الأداة الحاضرة، والثاني إن أزلت الأداة المحضونة من الأداة الحاضرة. أخيراً، الحدث PropertyChanged يتم تنفيذه في كل مرة تقوم بتغيير احد خصائص الأداة، يرسل هذا الحدث مع وسيطته الخاصية PropertyChangedEventArgs والتي تمثل اسم الخاصية التي تم تعديلها.

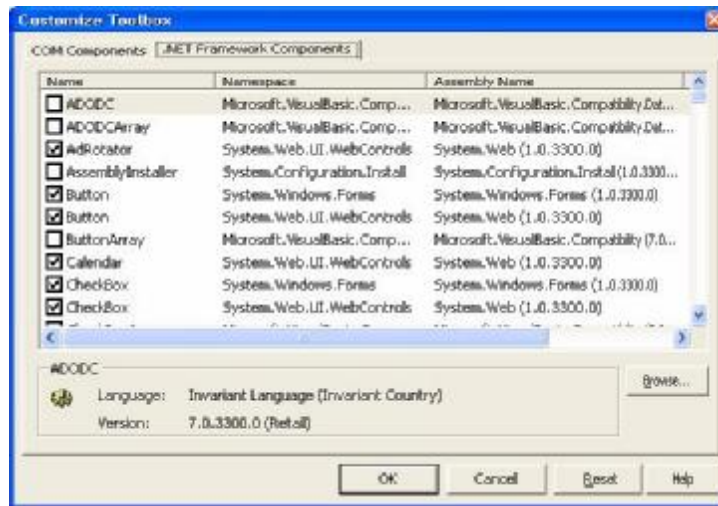
عرض سريع للأدوات

توجد العشرات من الأدوات التي تأتي بشكل ابتدائي مع رزمة Visual Studio .NET تجدها في نافذة صندوق الأدوات (شكل 15-5) والذي تصل اليه باختيار الامر Toolbox من قائمة View، اعلم ان الأدوات في شريط الأدوات لا تظهر الا ان كانت نافذة مصمم النماذج Form Desinger مفتوحة وظاهرة امام عينك.



شكل 14-5: نافذة صندوق الأدوات Toolbox.

يمكنك تحديد هذه الأدوات والبدء في رسمها على جبهة نافذة مصمم النماذج، كما تستطيع حذف أو إضافة أدوات جديدة بالضغط بزر الفأرة الايمن على صندوق الأدوات، واختيار الامر Customize ToolBox من القائمة المنبثقة، ليظهر لك صندوق الحوار Customize Toolbox (شكل 14-6).



شكل 14-6: صندوق الحوار Customize Toolbox.

يحتوي صندوق الحوار Customize Toolbox على خانتي تبويب Tab، الاول تعرض لك كافة أدوات التحكم ActiveX Controls والتي تعتمد في بنيتها التحتية على تقنية COM القديمة، ومعظم الأدوات المعتمدة على تقنية COM لا تعمل بشكل صحيح مع تقنيتنا الجديدة .NET. بالنسبة لخانة التبويب الثانية NET Framework Component. تعرض لك أدوات التحكم الخاصة Custom Controls والذي قد يكون احد مصمميه انت -كما ستري لاحقا. في هذا القسم من الفصل سنحتك مع معظم الأدوات التي ظهرت لك بشكل ابتدائي في صندوق الأدوات Toolbox لحظة انشاء مشاريع من النوع Windows Application، لنعرض لك ابرز خصائصها، طرقها، وأحداثها. وكالعادة، يمكنك العودة إلى وثائق ومستندات .NET Documentation إن أردت المزيد من التفاصيل حول أعضاء هذه الأدوات، حيث اني لن اعرضها هنا بالا بشكل سريع ومختصر.

الأداة Label

تستخدم هذه الأداة البسيطة في عرض النصوص على النوافذ، يمكنك محاذاة النص باستخدام الخاصية `TextAlignment` بإسناد قيمة لها من 6 قيم (شكل 14-1)، كما تحتوي على الخاصيتين `PreferredWidth` و `PreferredHeight` التي تعود بأفضل حجم يناسب الأداة اعتماداً على نوع وحجم الخط المستخدم.

توجد الخاصية `FlatStyle` التي تمكنك من تغيير شكل الحد الخارجي للأداة، كما يمكنك عرض صورة في الأداة عن طريق الخاصية `Image` ويمكنك التحكم في موقع الصورة عن طريق الخاصية `ImageAlign`.

أسند القيمة `True` إلى الخاصية `UseMnemonic` لتمكن المستخدم من نقل التركيز إلى الأداة التي تلي أداة `Label` الحالية في ترتيب `TabIndex` عند قيامه بالضغط على المفتاح `[Alt]` و الحرف الذي يتبع للحرف `&` في الخاصية `Text`، مع العلم أنه سيظهر خط سفلي تحت الحرف الذي يلي حرف `&`. وإن أردت عرض الحرف `&` على جبهة الأداة في هذه الحالة، اكتبه مرتين متتاليتين `&&`.

الأداة LinkLabel

الأداة `LinkLabel` نسخة مطورة من الأداة `Label` السابقة، يمكنك من وضع روابط كالروابط الموجودة في صفحات `HTML`، بحيث تتمكن من كتابة شيفرات ردة فعل على هذه الروابط. توجد طريقتين لوضع الأداة، الطريقة السريعة ممكنة في وقت التصميم لتعرض لك رابط واحد في كامل النص تحدد في الخاصية `LinkArea`، فلو كان قيمة الخاصية `Text` هي "مرحباً بكم في شبكة المطورون العرب" يمكنك وضع قيمة البداية 13 والحجم 19 في الخصائص `Start` و `Length` التابعة للخاصية `LinkArea`.

أما الطريقة الثانية فهي ممكن وقت التنفيذ فقط، بحيث تتمكن من وضع مجموعة من الروابط في نفس نص الأداة عن طريق الخاصية `Links` والتي تمثل مجموعة `Collection`:

```
LinkLabel1.Text = "يمكنك الضغط هنا أو الضغط هنا"
LinkLabel1.Links.Add(12, 3, "Link1")
LinkLabel1.Links.Add(25, 3, "Link2")
```

سواء استخدمت الأولى أو الثانية، تستطيع الاستفادة من الحدث `LinkClicked` لكتابة شيفرات ردة الفعل عند النقر على أحد روابط الأداة:

```
Private Sub LinkLabel1_LinkClicked(ByVal sender As System.Object, _
    ByVal e As System.Windows.Forms.LinkLabelLinkClickedEventArgs) _
    Handles LinkLabel1.LinkClicked

    Select Case e.Link.LinkData()
        Case "Link1"
            MsgBox("تم نقر الرابط الاول")
        Case "Link2"
            MsgBox("تم نقر الرابط الثاني")
    End Select

End Sub
```

الأداة TextBox

تعتبر الأداة TextBox الوسيلة المثلى لقص المدخلات من المستخدمين، واستخدامها يعتبر جزءاً لا يتجزأ من أي نافذة نموذج موجه لاستقبال المعلومات والبيانات، النص الظاهر في وسط الأداة هو نفس النص الموجود في الخاصية Text، وعند أي تغيير لهذه الخاصية سيتم تنفيذ الحدث TextChanged. كما يمكنك منع المستخدم من تحرير الأداة بإسناد القيمة True إلى الخاصية ReadOnly - رغم أن فرصة تغيير محتوياتها برمجياً ممكنة. يمكنك تحديد نص معين من النص الظاهر وسط الأداة عن طريقة الخاصيتين SelectionStart و SelectionLength، الأولى تحدد فيها نقطة البداية والثانية عدد الحروف، الشيفرة التالية تحدد جميع الحروف في أداة النص:

```
TextBox1.SelectionStart = 0
TextBox1.SelectionLength = TextBox1.Text.Length
```

إن كان النية تحديد جميع الحروف في أداة النص، فالشيفرة التالية يمكنك تقليصها إلى سطر واحد وذلك باستدعاء الطريقة SelectAll() التي تحدد كامل النص، كما يمكنك معرفة النص المحدد عن طريق الخاصية SelectedText.

المزيد أيضاً، أرسل قيمة مع الطريقة AppendText لإضافة نص في نهاية النص الحالي، كما توجد الخاصية AutoSize لتحجيم الأداة لتناسب مع حجم ونوع الخط بشكل تلقائي. ويمكنك استدعاء الطريقة Undo() لإعادة القيمة الأخيرة في أداة النص، تشترط هذه الطريقة أن تكون قيمة الخاصية CanUndo هي True، وعليك التحقق من أن الأداة قد تم تعديلها فعلاً عن طريقة الخاصية Modified.

يمكنك تحديد العدد الأقصى للحروف الممكن إدخالها في أداة النص عن طريقة الخاصية MaxLength، كم تستطيع إسناد حرف إلى الخاصية PasswordChar إن أردت استخدامه ليتم عرضه مهما كانت الحروف (يفيدك حرف النجمة * لمحاكاة كلمات المرور).

ملاحظة

إن استخدمت الخاصية PasswordChar، فإن ذاكرة أداة النص TextBox كفيلة بإلغاء أوامر النسخ Copy والقص Cut التابعة للقائمة المنبثقة والتي تظهر إن قام المستخدم بالضغط بزر الفأرة الأيمن على الأداة. أما إن أنشأت قوائم نسخ ولصق بنفسك، فذاكرتك هي المسئول الاول والاخير عن إلغائها.

عند الحديث عن الحروف الإنجليزية، يمكنك إسناد القيمة Upper للخاصية CharacterCasing حتى تجعل جميع الظهور كبيرة Capital واسناد القيمة Lower لتتحول الحروف إلى صغيرة.

أما الحديث عن الحروف العربية، فدعني اذكرك بان الحروف المتداخلة (مثل: لا، لأ، لا... الخ) هي عرفين مستقلين، كذلك الحال مع علامات التشكيل والتتوين (َ، ُ، ِ)، لذلك ضع استقلالية الحروف في الاعتبار عند التعامل مع خصائص الأداة.

أداة متعددة السطور:

إسناد القيمة True للخاصية Multiline يحول الأداة إلى أداة متعددة السطور Multi-Line TextBox بحيث تمكن المستخدم من كتابة سطور متعددة في نفس الأداة.

تستطيع إسناد القيمة True أيضا إلى الخاصيتين AcceptsReturn و AcceptsTab إن أردت التعامل مع المفاتيح [Enter] و [Tab] كحروف، فكما تعلم ان المفتاح [Enter] يقوم بتنفيذ زر الاوامر الافتراضي (ان وجد) في نافذة النموذج، والمفتاح Tab ينقل التركيز إلى الأداة التالية في الخاصية TabIndex.

ملاحظة

الزر الافتراضي Default Button هو الزر الذي عليه حد إضافي خارجي يتم تنفيذه بمجرد الضغط على المفتاح [Enter] من قبل المستخدم على نافذة النموذج - كما ستري في الفقرة القادة الأداة Button. مع ذلك، ان لم يكن هناك زر افتراضي في النافذة فلست بحاجة إلى استخدام الخاصية AcceptsReturn.

يمكنك إضافة اشرطة تمرير للأداة متعددة السطور عن طريق الخاصية ScrollBars، مع العلم ان شريط التمرير الأفقي H ScrollBar لن يظهر الا ان كانت قيمة خاصية الالتفاف WordWrap تساوي False .

يمكنك الاستفادة من الخاصية Lines والتي تعود بمصفوفة تمثل قيم السطور المختلفة في الأداة، كما تستطيع استدعاء الطريقة ScrollToCaret() لتحريك أشرطة التمرير بحيث تظهر لك الجزء المحدد من النص.

التحقق من المدخلات:

الحديث Validating و Validated سيفيدانك كثيرا عند التعامل مع الأداة TextBox، وذلك لانهما يعتبران انسب مكان للتحقق من المدخلات في أداة النص.

سيناريو تنفيذ الحدثان يكون كالتالي: عندما ينتقل التركيز من الأداة X إلى الأداة Y، سيتم التحقق من الخاصية CausesValidation لكلا الأداةين، ان كانت قيمة الخاصيتين False فلن يحدث شيء، وان كانت قيمة الخاصية True فسيتم تنفيذ الحدث Validating التابع للأداة X، وان قمت باسناد القيمة True إلى الخاصية Cancel التابعة لوسيلة الحدث Validating:

```
Private Sub X_Validating(ByVal sender As Object, _
    ByVal e As System.ComponentModel.CancelEventArgs) _
    Handles X.Validating

    If X.Text = "" Then e.Cancel = True
End Sub
```

فسيعود التركيز إلى الأداة X ولن يتم تنفيذ الحدث Validated التابع لنفس الأداة (ولا حتى الحدث LostFocus أيضا)، اما ان لم تفعل شيئا للخاصية Cancel السابقة، فسيتم تنفيذ الحدث Validated ومن ثم نقل التركيز إلى الأداة Y.

الأداة Button

استخدام هذه الأداة معروف وسهل جدا حتى لمستخدمين Windows العاديين، وهو زر يتم ضغطه لتنفيذ أوامر معينة. لا يوجد الكثير لأخبرك به حول هذا الزر سواء وجود خاصيتين تابعة لنافذة النموذج تؤثران تأثيرا بسيطاً على هذا الزر هما `AcceptButton` و `CancelButton`، تحدد في الأولى الزر الذي تود رسم حوله حد اضافي يخبر المستخدم بان المفتاح `[Enter]` يؤدي إلى تنفيذ هذا الزر، والخاصية الثانية مرافقة للمفتاح `[Esc]`، يمكنك تعديل هذه الخصائص لنافذة النموذج وقت التصميم من نافذة الخصائص، أو وقت التنفيذ بهذه الشيفرة:

```
Me.AcceptButton = Button1
Me.CancelButton = Button2
```

الأداة CheckBox

تملاً هذه الأداة اغلب تطبيقات Windows، يمكنك تحديد ما اذا كانت الأداة مختارة باسناد القيمة `True` إلى الخاصية `Checked` و `False` لالغاء الاختيار، عند اسنادك للقيمة `True` للخاصية السابقة، يمكنك تحديد القيمة `Indeterminate` للخاصية `CheckState` والتي تماثل ما بين نعم و لا.

عندما يقوم المستخدم بالنقر على الأداة سيتم عكس قيمة خاصيتها `Checked` بشكل تلقائي، مع ذلك تستطيع منع هذا التغيير باسناد القيمة `False` إلى الخاصية `AutoCheck`، لتحصر المسؤولية عليك في كتابة الشيفرات البرمجية واللازمة لتغيير قيمة الخاصية `Checked`. الخاصية `CheckAlign` مثل الخاصية `TextAlign` (شكل 14-1) تماماً، ويكن الفرق في ان الأولى خاصة بموقع رمز المربع ☒ فقط، بينما الثانية فخاصة بالنص المرافق لرمز المربع.

كما ذكرت قبل قليل، عندما يقوم المستخدم بالنقر على الأداة فسيتم عكس قيمة الخاصية `Checked`، ولكن عن إسناد القيمة `True` للخاصية `ThreeState` فلن يتم عكس قيمة الخاصية `Checked` إلى كل نقرتين، النقرة الأولى تجعل الخاصية `Checked` هي `True` والثانية تضيف القيمة `Indeterminate` للخاصية `CheckState`، اما النقرة الثالثة فتعكس قيمة الخاصية `Checked` وهكذا ...

ملاحظة

لن تعمل الخاصية ThreeState إلا إن كانت قيمة الخاصية AutoCheck هي True.

اخيرا، تحتوي الأداة CheckBox على الحدث CheckChanged والذي يتم تنفيذه ان تم تغيير قيمة الخاصية Checked أو CheckState.

الأداة RadioButton

وجه الشبه بين هذه الأداة والأداتين التي قبلها (CheckBox و Button) هو ان كلاهم مشتق وراثيا من الفئة ButtonBase والتي تحتوي على خصائص اضافية كـ FlatStyle لتحديد شكل ثلاثي الأبعاد 3D والخاصية Appearance التي تمكنك من استخدام شكل الزر Button مع الأداتين CheckBox و RadioButton.

يمكنك إسناد القيمة True للخاصية Checked التابعة لهذه الأداة لاختيارها، مع العلم ان باقي الأدوات في نفس المجموعة (نفس الأدوات المحضونة في الأداة الحاضنة) سيتم إسناد القيمة False لخصائصها Checked.

الأداة ListBox

إن أردت التعامل مع الأداة بشكل عام، فانك تستخدم كائناتها المنشئ من الفئة ListBox، اما إن أردت التعامل مع العناصر الموجودة في الأداة، فوجه أنظارك إلى الخاصية Items والتي عبارة عن مجموعة Collection تمثل عناصر الأداة ListBox. بما ان الخاصية Item عبارة عن مجموعة Collection، فهي تحتوي على الواجهة ICollection فيمكنك استخدام الطرق التقليدية لإضافة، حذف، والاستعلام عن العناصر (كـ Add(), Insert(), Clear(), Count... الخ):

```
' إضافة عناصر '
Dim counter As Integer

For counter = 1 To 10
    ListBox1.Items.Add(counter)
Next

' حذف جميع العناصر
ListBox1.Items.Clear()
```

```
' الاستعلام عن العناصر
Dim item As Object
For Each item In ListBox1.Items
    ...
Next
```

انظر أيضا

يمكنك مراجعة الفصل السادس **الفئات الأساسية** لمزيد من التفاصيل حول الواجهة `ICollection`.

العناصر التي تضيفها لا يشترط ان تكون بيانات ابتدائية (Primitave Types) كـ `String`، `Integer`، `Double`... بل يمكن ان تشكل كائنات لفئات تعرفها بنفسك:

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

    Sub New(ByVal name As String, ByVal age As Integer)
        Me.Name = name
        Me.Age = age
    End Sub
End Class

...

With ListBox1.Items
    .Add(New Person("تركي", 99))
    .Add(New Person("عباس", 3000))
End With
```

السؤال الذي يطرح نفسه بقوة الان، ما هو النص الذي سيظهر في عناصر الأداة `ListBox`، والجواب هو اسم الفئة الذي تعود به الطريقة `ToString()` والتابعة للكائن. نستنتج من هذا، أننا نستطيع تحديد القيمة التي نريدها بفضل اعادة القيادة `Overriding`:

```
Class Person
    ...
    Overrides Function ToString() As String
        Return Me.Name
    End Function
End Class
```

يستطيع المستخدم تحديد أكثر من عنصر من عناصر أداة ListBox عن طريقة الخاصية SelectionMode والتي تسند لها اما القيمة MultiSimple أو القيمة MultiExtended (على المستخدم استخدام المفاتيح [Ctrl] أو [Shift] في حالة إسناد القيمة الثانية)، كما يمكن للمبرمج من الاستعلام عن جميع العناصر المحددة عن طريقة الخاصية SelectedItems:

```
Dim x As String
For Each x In ListBox1.SelectedItems
    ...
Next
```

وعلى ذكر التحديد، يمكن للمبرمج أيضا من تحديد/الغاء تحديد العنصر برمجيا باستدعاء الطريقة SetSelected()، والتي ترسل معها القيمة True لتحديد العنصر أو False لالغاء التحديد:

```
ListBox1.SetSelected(0, True)
ListBox1.SetSelected(1, False)
```

اخيرا، عند إضافة مجموعة كبيرة من العناصر، فينصح دائما باستدعاء الطريقة BeginUpdate() قبل إضافة العناصر والطريقة EndUpdate() بعد إضافتها، وذلك تمنع الأداة من اعادة رسمها في كل مرة تضيف عنصر جديد مما يزيد سرعة الإضافة أضعاف المرات:

```
Dim counter As Integer

ListBox1.BeginUpdate()
For counter = 0 To 10000
    ListBox1.Items.Add(counter)
Next
ListBox1.EndUpdate()
```

الأداة CheckedListBox

الأداة CheckedListBox هي نسخة محسنة من الأداة ListBox وهي مشتقة وراثيا منها، فكل ما ذكرته في السطور السابقة يطبق على هذه الأداة باستثناء الخاصية SelectionMode حيث لن تتمكن من استخدامها كما فعلت مع الأداة ListBox، وذلك ان طريقة تحديد العناصر تعتمد على أدوات شبيهة بالأداة CheckBox، اما إن أردت معرفة العناصر التي تم تحديدها فاستخدم الخاصية SelectedItems عوضا عن الخاصية SelectedItems.

الأداة ComboBox


الأداة ComboBox ما هي الا أيضا نسخة محسنة من الأداة ListBox السابقة وتحتوي على أداة TextBox، ولكنها ليست مشتقة منها، مع ذلك معظم الطرق والخصائص التابعة للأداة ListBox والأداة TextBox مدعومة في الأداة ComboBox أيضا.

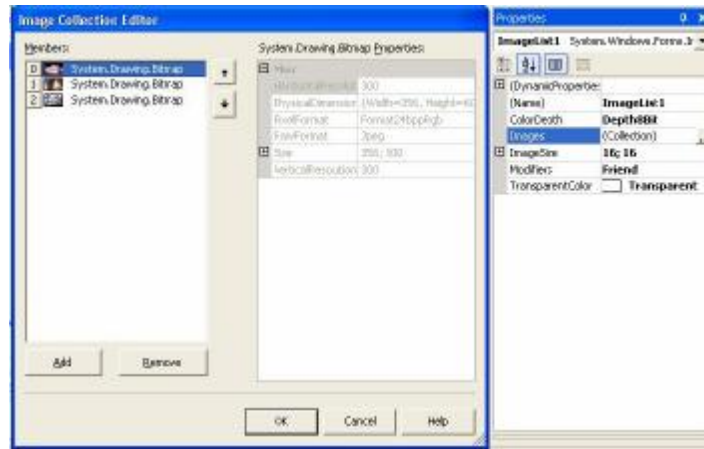
يمكنك تغيير شكل الأداة ComboBox عن طريق الخاصية DropDownStyle والتي تكون قيمة من ثلاث قيم هي: DropDown، Simple، و DropDownList، في الاولى يمكنك المستخدم من تحرير النص في خانة النص اما الثانية فلا، وبالنسبة للقيمة الثالثة فهي تمكن المستخدم من تحرير النص ولكنها تظهر عناصر الأداة بشكل مبثني.

ان اخترت القيمة الاولى أو الثانية، يمكنك عرض قائمة عناصر الأداة في اي وقت برمجيا باسناد القيمة True إلى الخاصية DroppedDown.

الأداة ImageList

تستخدم هذه الأداة كمحفظة أو حاوية للصور التي تود عرضها على الأدوات الاخرى، صحيح ان معظم الأدوات يمكنك وضع قيم صور لها مباشرة عن طريق خاصيتها Image، الا ان استخدام الأداة ImageList سيوفر عليك مساحة عند حفظ الصور المتكررة، ليس هذا فقط بل ان بعض الأدوات (كـ TreeView و ListView) لن تتمكن من عرض رموز على عناصرها الا ان وجدت أداة ImageList على جبهة نافذة النموذج.

يمكنك إضافة وحذف الصور في الأداة ImageList وقت التصميم، وذلك عن طريق الخاصية Images، انتقل إلى نافذة الخصائص واضغط على الزر  المقابل للخاصية ليظهر لك صندوق حوار بعنوان Image Collection Editor (شكل 14-7) يمكنك من إضافة الصور.



شكل 14-7: إضافة صور للأداة ImageList

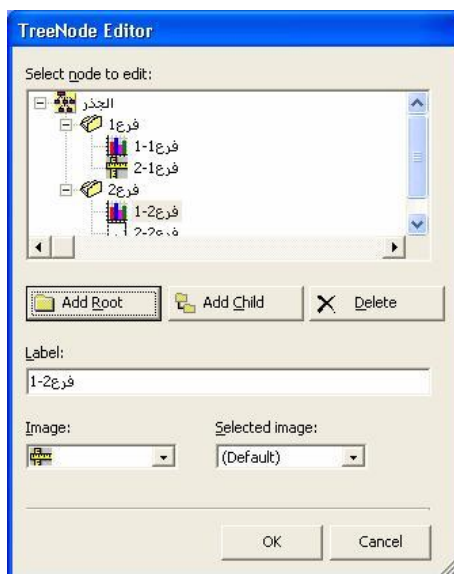
ملاحظة

الخاصية Item عبارة عن مجموعة Collection، فهي تحتوي على الواجبة ICollection لذلك، يمكنك استخدام الطرق التقليدية لإضافة، حذف، والاستعلام عن الصور (كـ Add()، Insert()، Clear()، Count())... الخ) وقت التنفيذ.

الأداة TreeView

تمتلك الأداة TreeView من عرض العناصر على شكل شجري كما تعرض مجلدات مستكشف النظام Windows Explorer، تتطلب هذه الأداة أداة ImageList إن أردت عرض صور ورموز على عناصر الأداة، يمكنك تحديد ارفاق أداة ImageList إلى الأداة TreeView عن طريق الخاصية ImageList والتابعة للأداة TreeView.

العناصر التي تضيفها إلى الأداة تسمى Nodes، يمكنك تحريرها وقت التصميم عن طريق الخاصية Nodes والتي تمثل مجموعة Collection للعناصر، اضغط على الزر المرافق للخاصية في نافذة الخصائص ليظهر لك صندوق الحوار Tree Node Editor (شكل 14-8).



شكل 14-8: تحرير عناصر Node الأداة TreeView.

ستستخدم الخاصية Nodes دائما في شيفراتك البرمجية إن أردت التعامل مع العناصر (كما تفعل مع الخاصية Items التابعة للأداة ListBox) وقت التنفيذ، الا انك ستتشئ كائن من النوع TreeNode عند إضافة العنصر بالطريقة Add() أو Insert():

```
TreeView1.Nodes.Insert(0, New TreeNode("عنصر جذري"))
TreeView1.Nodes(0).Nodes.Add(New TreeNode("عنصر فرعي"))
```

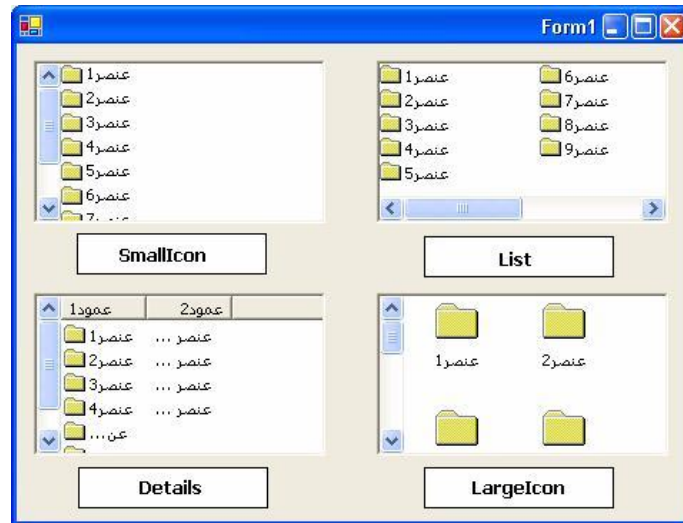
من خصائص الأداة TreeView الخاصية ShowLines التي تظهر خطوط تربط العناصر الفرعية بالعناصر الجذرية، الخاصية ShowPlusMinus التي تظهر علامات الزائد (+) والناقص (-) للعناصر الجذرية، والخاصية Indent التي تحدد فيها المسافة بين العنصر الجذري وحد الأداة الأيسر.

ملاحظة

الخاصية RightToLeft تحاكي أشرطة تمرير الأداة بالاتجاه الصحيح ولكنها لا تشمل عناصر الأداة Nodes، حيث ستضل من اليسار إلى اليمين. إن أردت تحويلها إلى الاتجاه العربي عليك استخدام تقنية المرآة Mirroring كما ستري لاحقاً.

الأداة ListView

تمتلك الأداة ListView من عرض عناصر على شكل ايقونات كما يفعل سطح المكتب Desktop ومستكشف النظام Windows Explorer. حدد في الخاصية View اسلوب من اربعة اساليب لعرض عناصر الأداة هي: List، SmallIcon، Details، و LargeIcon (شكل 14-9).



شكل 14-9: الاساليب المختلفة لعرض عناصر الأداة في الخاصية View.

تتطلب الأداة ListView اداتين من النوع ImageList لعرض صور في عناصر الأداة (يمكنك الاعتماد على أداة ImageList إن أردت)، حددها في الخاصية SmallImageList الأداة ImageList التي تود عرض رموزها في الحالات الثلاث للخاصية View، وحدد في الخاصية

LargeImageList الأداة ImageList التي تستخدم لعرض الرموز الكبيرة (الحالة LargeIcon للخاصية View).

التعامل مع الأداة ListView هو مثل التعامل مع الأداة TreeView السابقة، ولا يوجد داعي للتكرار، فإن كانت الأداة TreeView تعتمد على المجموعة Nodes لعرض عناصرها، فإن الأداة ListView تعتمد على المجموعة Items لعرض عناصرها.

إن كان أسلوب العرض للأداة ListView في الخاصية View هو Details، فتستطيع الاعتماد على المجموعة Columns لتحديد الأعمدة، وفي هذه الحالة ستستخدم المجموعة SubItems والتابعة لكل عنصر من عناصر المجموعة Items لتوزيع بيانات العنصر في الحقول (الأعمدة) المختلفة للأداة.

الأداتان StatusBar وToolBar

في أغلب الأحوال، تستخدم كلا الاداتين StatusBar و ToolBar في نافذة النموذج، الأداة الاولى تعرض لك شريط أدوات تستطيع تحرير ازراره وقت التصميم عن طريق الخاصية Buttons، والأداة الثانية تعرض لك سطر حالة يمكنك تحرير المربعات التي فيه عن طريق خاصيته Panels.

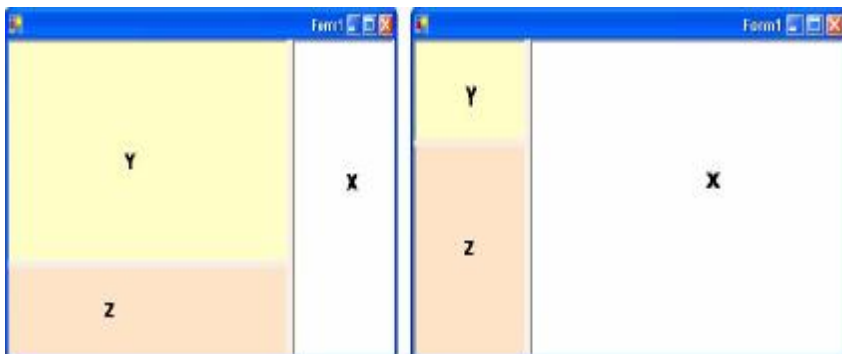
كلا الاداتين تتطلبان كائن من النوع ImageList لعرض الرموز على عناصرها، وكلا الخاصيتين Buttons و Panels تحتويان على الواجهة ICollection، لذلك تستطيع التعامل مع عناصرهما برمجيا كما تفعل مع المجموعة Nodes (لأداة TreeView) والمجموعة Items (للاداتين ListBox و ListView)، والاختلاف لا يتعدى نوعية واسماء الخصائص والطرق التي تناسب كل أداة.

ملاحظة

حتى تتمكن من رؤية مربعات Panels والخاصة بسطر الحالة في الأداة StatusBar، عليك إسناد القيمة True للخاصية ShowPanels.

الأداة Splitter

توفر الأداة Splitter على نفسك كتابة عشرات الأسطر من الشيفرات المصدرية والتي تتعلق باعطاء قابلية للمستخدم بتحجيم الأدوات (كما تفعل نوافذ مستكشف النظام Windows Explorer)، فكل ما يتطلبه منك الأداة خطوات بسيطة بالفارة وستجعل أدواتك قابلة للتحجيم من قبل المستخدم (شكل 14-10).



شكل 14-10: تحجيم الأدوات وقت التنفيذ باستخدام الأداة Splitter.

تتعامل الأداة Splitter مع الأدوات المُحاذاة بالخاصية Dock ولن تستطيع تعلم استخدام الأداة Splitter الا بتطبيق عملي عليها، اضع أداة X في نافذة النموذج وقم باسناد القيم Right لخاصيتها Dock (لن يتم محاذاة الأداة في الجزء الايمن من النافذة)، اضع الان أداة Splitter وغير خاصيتها Dock إلى Right، اضع أداة Y واسند القيمة Top لخاصيتها Dock، اضع أداة Splitter اخرى بنفس القيمة Top لخاصيتها Dock، واخيرا اضع أداة Z واسند القيم Fill لخاصيتها Dock.

نفذ البرنامج وحاول تغيير حجم الأدوات يمينا ويسارا، فوق وتحت (شكل 14-10).

أدوات صناديق الحوار الشائعة

توجد ست أدوات يمكنك استخدامها لعرض مجموعة من صناديق حوار نظام Windows الشائعة، كصندوق حوار فتح، حفظ، الطابعة... الخ. التعامل مع هذه الأدوات يتشابه إلى حد كبير. يكفي إضافة نسخة واحد من كل أداة على نافذة النموذج حتى تتمكن من فتح صندوق الحوار أكثر من مرة مع تغيير خصائص العرض وبياناته.

الأداة OpenFileDialog:

تستخدم هذه الأداة لعرض صندوق حوار **فتح Open**، يبدأ بالخاصية Filter لتحديد نوع الملفات التي تود عرضها، كما يمكنك إسناد القيمة True إلى الخاصية MultiSelect لتمكين المستخدم من اختيار أكثر من ملف، الخاصية ShowReadOnly تظهر أداة من النوع CheckBox على صندوق الحوار بعنوان ReadOnly (يمكنك معرفة ما إن قام المستخدم بتحديد هذا الاختيار عن طريق الخاصية ReadOnlyChecked)، الخاصية CheckFileExists التي تجبر المستخدم على اختيار ملف موجود، والخاصية InitialDirectory التي تحدد المسار الابتدائي لصندوق الحوار.

بعد إسناد قيم للخصائص الآتية، يمكنك استدعاء الطريقة ShowDialog() لفتح صندوق الحوار، ستعود الطريقة بالقيمة DialogResult.OK إن تم الضغط على زر Open والقيمة DialogResult.Cancel إن كائن الضغط على الزر Cancel:

```
With OpenFileDialog1
    .CheckFileExists = True
    .Filter = "*.*)" & "كل الملفات"
    .FilterIndex = 2
    .InitialDirectory = "C:\\"

    If .ShowDialog = DialogResult.OK Then
        ...
    Else
        ...
    End If
End With
```

الأداة SaveFileDialog:

تعرض لك هذه الأداة صندوق حوار **حفظ Save** وهي تشابه إلى حد كبير الأداة OpenFileDialog السابقة، باستثناء بعض الخصائص التي لا تتناسب مع غرض ومهمة صندوق الحوار (كالخاصية ShowReadOnly). المزيد أيضاً، أسند القيمة True إلى الخاصية OverwritePrompt إن أردت اظهار رسالة تنبيه في حال ما تم اختيار ملف موجود.

الأداة ColorDialog:

تعرض هذه الأداة صندوق حوار الألوان لتمكين المستخدم من اختيار اللون بطريقة أفضل، ويعتبر استخدام هذه الأداة سهل جدا حيث سيكون جل تركيزك على الخاصية Color:

```
With ColorDialog1
    If .ShowDialog = DialogResult.OK Then
        Me.BackColor = .Color
    End If
End With
```

الأداة FontDialog:

اما الأداة FontDialog فتعرض لك صندوق حوار الخطوط Fonts، تحتوي على مجموعة من الخصائص الإضافية كالخاصية ShowColor لتظهر قائمة الألوان (يمكنك الاستعلام عن اللون المختار عن طريق الخاصية Color). يمكنك فتح صندوق الحوار هذا بنفس الطرق السابقة:

```
With FontDialog1
    .ShowColor = True
    If .ShowDialog = DialogResult.OK Then
        TextBox1.Font = .Font
        TextBox1.ForeColor = .Color
    End If
End With
```

لإعطاء مرونة لمستخدمي برامجك، لما لا تمكنهم من تحديد الخطوط ورؤية النتائج دون إغلاق صندوق الحوار وذلك بالضغط على الزر Apply - عوضا عن OK، يتم ذلك بإسناد القيمة True إلى الخاصية ShowApply، ويمكن كتابة ردة الفعل عند الضغط على هذا الرز في الحدث Apply:

```
Private Sub FontDialog1_Apply(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles FontDialog1.Apply

    TextBox1.Font = FontDialog1.Font
    TextBox1.ForeColor = FontDialog1.Color
End Sub
```

الأداة **PrintDialog**:

هذه الأداة تعرض لك صندوق حوار اختيار الطابعة، من أهم خصائصها الخاصية **PrinterSettings** والتي عبارة عن كائن من النوع **PrinterSettings** يحتوي على جميع الإعدادات الموجودة في صندوق حوار الطابعة، الشيفرة التالية توضح لك طريقة استخدامه (الفئة **PrinterSettings** مشمولة في مجال الاسماء **(System.Drawing.Printing)**):

```
With PrintDialog1
    .AllowSomePages = True

    .PrinterSettings = New System.Drawing.Printing.PrinterSettings()

    If .ShowDialog = DialogResult.OK Then
        With .PrinterSettings
            MsgBox(.PrinterName)
            MsgBox(.FromPage)
            MsgBox(.ToPage)
        End With
    End If
End With
```

الأداة **PageSetupDialog**:

ان كان الأداة **PrintDialog** تعرض لك صندوق حوار اختيار الطابعة، فان الأداة **PageSetupDialog** تعرض لك صندوق حوار اعدادات صفحة الطباعة **Page Setup**، والتي يحدد فيها المستخدم معلومات تفصيلية (كالهوامش، حجم الورق، الطباعة الأفقية أو العمودية... الخ)، يمكنك الحصول على هذه المعلومات عن طريق الخاصية **PageSettings** والتي تعود بكائن من النوع **(System.Drawing.Printing.PageSettings)**:

```
With PageSetupDialog1
    .AllowPaper = True

    .PageSettings = New System.Drawing.Printing.PageSettings()

    If .ShowDialog = DialogResult.OK Then
        With .PageSettings
            MsgBox(.PaperSize.Height & "x" & .PaperSize.Width)
        End With
    End If
End With
```

أدوات المزودات

أدوات المزودات **Provider Controls** هي مجموعة من الأدوات التي تضيف خصائص جديدة على جميع الأدوات الموجودة في نافذة النموذج لتطويرها وإضافة امكانية جديدة عليها. في هذه الفقرة ساعرض لك اداتين من هذا النوع مع العلم انه يمكنك تطوير أدوات مزودات خاصة بك.

الأداة ToolTip:

تمتلك الأداة ToolTip من عرض مستطيل التلميح على الأداة (شكل 11-14)، واستخدامها سهل جدا، فكل ما هو مطلوب منك إضافة نسخة من الأداة ToolTip على النموذج، وبذلك تضيف الخاصية ToolTip on ToolTip1 لكل أداة موجودة على نافذة النموذج، أسند قيمة حرفية لهذه الخاصية الجديدة في كل أداة ليتم عرضها كتلميح ان ظل مؤشر الفأرة فترة من الوقت دون تحريك على سطح الأداة.



شكل 11-14: مستطيل التلميح ToolTip.

ملاحظة

جميع أدوات المزودات Provider Controls تضيف خصائص جديدة على الأدوات بالصيغة "اسم الأداة on اسم الخاصية".

المزيد أيضا، تستطيع التحكم في الفترة من الوقت المطلوبة لظهور التلميح عن طريق مجموعة من الخصائص AutoPopDelay، ReshowDelay، و AutomaticDelay رغم اني لا احبذ لك تعديلها.

الأداة HelpProvider:

الأداة HelpProvider تعمل كحلقة وصل بين الفئة Help، بحيث تتمكنك من الاتصال بملف التعليمات وعرض ومحتوياته عندما يقوم المستخدم بالضغط على المفتاح [F1] عندما يكون التركيز على أداة معينة.

انظر أيضا

الفئة Help فئة خاصة بالتعامل مع ملفات التعليمات Help Files، الفصل القادم **مواضيع متقدمة** يعرض أمثلة لاستخدام هذه الفئة.

تحتوي الأداة HelpProvider على خاصية وحيدة هي HelpNamespace والتي تسند لها اسم ملف التعليمات MyHelpFile.CHM، وبعد وضعها على نافذة النموذج، ستضيف ثلاث خصائص على كل أداة من الأدوات هي: HelpNavigator، HelpKeyword، و HelpString.

الخاصية الأولى تحدد فيها الصفحة التي تود عرضها في ملف التعليمات، أو صفحة المحتويات Contents، أو الفهرس Index... الخ، والخاصية الثانية تحدد فيها الكلمة التي تود البحث عنها في قسم البحث من ملف التعليمات، أما الخاصية الأخيرة HelpString فتظهر مستطيل تلميح (شكل 14-11) يتم عرضه بمجرد الضغط على المفتاح [F1] على الأداة (لأبد مسح قيمة الخاصية HelpKeyword لتفعيل هذه الخاصية).

أدوات أخرى

في السطور السابقة عرضت ثلاث وعشرين أداة تستخدمها في معظم مشاريعك المبنية على Windows Forms بشكل مختصر، تبقى مجموعة إضافية من الأدوات التي يكفي ذكر اسمها والوظيفة التي تقوم بها.

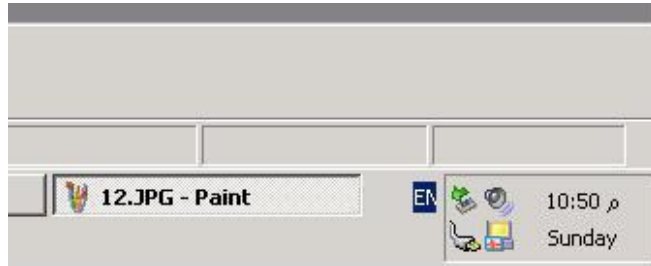
الأداة PictureBox أداة بسيطة تمكنك من وضع صور عليها في خاصيتها Image، كما تستطيع تحجيم الصورة لتغطي كامل الأداة أو تحجيم الأداة لتغطي كامل الصورة عن طريق الخاصية SizeMode.

الاداتان Panel و GroupBox كلاهما من النوع الحاضن Container مثل نافذة النموذج، ابرز الفروق بينهما في دعم الخاصية AutoScroll حيث الثانية تفقر هذه الخاصية كما انك لا

تستطيع اخفاء حدودها، مع ذلك توجد ميزة ليست موجودة في الأداة الاولى وهي النص الظاهر في احد زوايا الأداة تحدد عن طريق الخاصية Text.

بالنسبة للادائين HScrollBar و VScrollBar فلا اعتقد انك ستكثر من استخدامها الا ان كنت تنوي تطوير أدوات Custom Controls خاصة بك، وذلك لان معظم الأدوات -التي ذكرناها في هذا الفصل- تحتوي على اشرطة تمرير. حدد اعلى واصغر قيمة لشريط التمرير في الخاصيتين Maximum و Minimum، وانتظر وقوع الحدث Scroll ان تم تحريك اشرطة التمرير لتختبر موقع مربع شريط التمرير في الخاصية Value.

لديك الأداة NotifyIcon والتي يمكنك من وضع رموز في صينية النظام System Tray كما تفعل اغلب برامج Windows الخدمية (شكل 14-12). حدد الرمز الذي تود عرضه في الخاصية Icon وقد تضيف مستطيل تلميح في الخاصية ToolTip، كما تستطيع عرض واخفاء الرمز من صينية النظام في اي وقت عن طريق الخاصية Visible. اما إن أردت ارفاق قائمة منبثقة مع ذلك الرمز، فحدد كائن القائمة المنبثقة في الخاصية ContextMenu.



شكل 14-12: الرموز في خانة صينية النظام System Tray.

الأداة TabControl يمكنك من وضع خانات التبويب Tab كما تفعل اغلب صناديق الحوار في تطبيقات Windows (شكل 14-13)، كل خانة تبويب من هذه الخانات تمثل كائن من النوع tabPage وهو كائن حاضن Container. يمكن تحرير خانة التبويب وقت التصميم عن طريق الخاصية TabPages التي تحتوي على الواجهة ICollection أيضاً، كما تتطلب الأداة ImageList حتى تضع الرموز في اعلى خانات التبويب. اخيراً، عليك استخدام تقنية المرأة حتى توجه رؤوس خانات التبويب إلى الاتجاه العربي.



شكل 13-14: ثلاث خانات تبويب للأداة TabControl.

إن أردت أداة مثل الأداة TextBox ولكنها تدمج تنسيقات مختلفة من الخطوط، الألوان، الاحجام، وحتى الصور، فقد نحتاج إلى الأداة RichTextBox والتي تحتوي على خصائص وطرق كثيرة جدا جدا تجد تفاصيلها في مستندات .NET Documentation، مبدئيا لست بحاجة إلى معرفة كل هذه الخصائص، حيث ان اغلبها مشابه إلى حد كبير خصائص الأداة TextBox العادية.

يمكنك تسهيل اختيار القيم العددية بدلا من كتابة الأرقام على المستخدم باستخدام الأداة NumericUpDown، كما تستطيع التسهيل عليه أكثر باختيار قيم التاريخ عن طريق الاداتين DateTimePicker و MonthCalendar. وان كانت المهام المنجزة طويلة، فيفضل عرض شريط نسبة مئوية للمستخدم باستخدام الأداة ProgressBar.

أخيرا، أداة الموقت Timer تمكنك من تنفيذ شيفرات في حدثها الوحيد Tick والذي يتم تنفيذه كل فترة معينة تحددها في خاصية الأداة Interval (وحدثها 0.001 ثانية)، يمكنك بدء تنفيذ الحدث باسناد القيمة True إلى الخاصية Enabled أو القيمة False لايقاؤه (لا تسند القيمة 0 إلى الخاصية Interval حتى تتقاضي وقوع استثناء Exception وقت التنفيذ).

تقنية المرآة

على مر العقود الأخيرة، واجه المبرمجون العرب معاناة كبيرة في تطوير واجهات استخدام تحمل معايير وسمات عربية خالصة، وإن كنت من المبرمجين المخضرمين، فستذكر أن بدايات معاناتنا كانت تحت أنظمة MS-DOS، حيث اضطررنا إلى استخدام أدوات تعريب كنافذة Nafitha، المساعد العربي Arabic Helper، وغيرها من أنظمة التعريب التي لا أذكر اسمائها. لست بصدد العودة إلى الوراء وذكر المصاعب المختلفة التي وقفت حاجزا للمطورين العرب، وكيف كانوا يتلهفون على العبارة "Right To Left" -فهو موضوع طويل، ولكن دعني أ تسارع في الأيام لنصل إلى نماذج Windows Forms (التابعة لتقنية NET. الحالية).

الخاصية RightToLeft

الفئة Control تحتوي على الخاصية RightToLeft، وبالتالي فإن جميع الأدوات -بما فيها نافذة النموذج- تشمل هذه الخاصية والتي تسند لها قيمة من ثلاث قيم (تابعة لتركيب من النوع Enum باسم RightToLeft):

الوظيفة	القيمة
قلب اتجاه الأداة إلى الاتجاه العربي (من اليمين إلى اليسار).	Yes
قلب اتجاه الأداة إلى الاتجاه الغربي (من اليسار إلى اليمين).	No
قلب اتجاه الأداة إلى نفس اتجاه الأداة الحاضنة.	Inherit

عند تغيير اتجاه الأداة إلى الاتجاه العربي، فلا يوجد أي تغيير تطلبه شيفراتك المصدريّة باستثناء الشيفرات التي تستخدم الخاصية TextAlign، حيث أن قيمة Right ستحاكي النص إلى جهة اليسار، والقيمة Left إلى جهة اليمين، لذلك يتحتم عليك وضع ذلك في عين الاعتبار عن تغيير اتجاه الأداة إلى الاتجاه العربي.

بالنسبة لأشرطة التمرير ScrollBars، فسيتم نقل شريط التمرير العمودي إلى الجهة المقابلة -أي يسار الأداة- إن كانت تحتوي على شريط تمرير عامودي، أما شريط التمرير الأفقي فسيضل مكانه ولكن عملية تحريك ستكون معاكسة.

المزيد أيضا، عند تغيير قيمة الخاصية RightToLeft سيتم تنفيذ الحدث RightToLeftChanged التابعة للفئة Control، وبالتالي فهو مدعوم من قبل جميع الأدوات.


عند تصميم نافذة نموذج بالشكل الغربي، فإن تنسيقات ومواقع الأدوات تكون مرتبة بأسلوب يتماشى مع طبيعة الاتجاه المستخدم. فمثلاً، الأزرار Buttons توضع في جهة اليمين، وأدوات Label توضع يسار الأدوات TextBox... الخ، وإن فكرت في جعل اتجاه هذه النافذة من اليمين إلى اليسار، عليك إعادة تحريك جميع الأدوات، فالأزرار ستضعها جهة اليسار، وأدوات Label ستكون يمين أدوات TextBox وغيرها، مع ذلك يمكنك جعل المسألة تتم تلقائياً وذلك بتعريف إجراء MirrorLocations:

```
Private Sub MirrorLocations(ByVal ctl As Control)
    Dim C As Control

    For Each C In ctl.Controls
        C.Location = New Point(C.Parent.ClientRectangle.Width _
            - C.Size.Width - C.Location.X, C.Location.Y)

        If C.Controls.Count > 0 Then
            MirrorLocations(C)
        End If
    Next
End Sub
```

كما اعتقد انه سيكون افضل مكان مناسب نستدعي الإجراء MirrorLocations منه هو الحدث RightToLeftChanged:

```
 Private Sub Form1_RightToLeftChanged(ByVal sender As Object, _
    ByVal e As System.EventArgs) Handles MyBase.RightToLeftChanged

    MirrorLocations(Me)
End Sub
```

ذكرت قبل بضعة سطور، أن قيم الخاصية TextAlign سيتم عكسها إن تم تغيير قيم الخاصية RightToLeft، مع ذلك لست بحاجة إلى تغييرها في الإجراء MirrorLocation السابق، حيث أن Windows Forms ذكية بما فيه الكفاية لتغيير قيمها. إلا أن ذكاء Windows Forms لا يشمل الخاصيتين Anchor و Dock، فهما لا يتغيران من ناحية تنسيقية -كالخاصية TextAlign، ولكن المستخدم قام بوضع قيمهم على أساس اتجاه التصميم، لذلك يفضل بعكس قيم Right و Left، ليكون الشكل النهائي للإجراء MirrorLocation:



```
Private Sub MirrorLocations(ByVal ctl As Control)
    Dim C As Control

    For Each C In ctl.Controls
        C.Location = New Point(C.Parent.ClientRectangle.Width _
            - C.Size.Width - C.Location.X, C.Location.Y)
        If CBool(C.Anchor And AnchorStyles.Left) Then
            C.Anchor = (C.Anchor Or AnchorStyles.Right) Xor
            AnchorStyles.Left
        ElseIf CBool(C.Anchor And AnchorStyles.Right) Then
            C.Anchor = (C.Anchor Or AnchorStyles.Left) Xor
            AnchorStyles.Right
        End If
        If C.Dock = DockStyle.Right Then

            C.Dock = DockStyle.Left
        ElseIf C.Dock = DockStyle.Left Then
            C.Dock = DockStyle.Right
        End If
        If C.Controls.Count > 0 Then
            MirrorLocations(C)
        End If
    Next
End Sub
```

قصور الخاصية RightToLeft

مع المزايا التي توفرها الخاصية RightToLeft لتطوير تطبيقات ذات واجهات عربية حقيقية، إلا أنها به بعض القصور الذي يظهر جلياً على بعض الأدوات وأولها نافذة النموذج Form، حيث أن استخدام الخاصية RightToLeft لا يؤدي إلا تغيير محاذاة عنوان النافذة (شكل 14-14)، ولا يعطي ان انطباع عربي حقيقة لاتجاه النافذة.



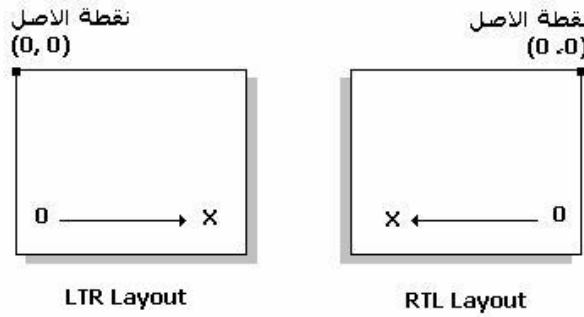
شكل 14-14: الخاصية RightToLeft لا تكفي لنافذة النموذج.

المزِيد أيضاً، الأدوات StatusBar، ProgressBar، Panel، ListView، TabControl، ToolBar، وTreeView لا تؤثر فيها قيمة الخاصية RightToLeft ولا تقوم بقلب اتجاه عناصرها إلى الاتجاه العربي. اما باقي الأدوات فتعمل الخاصية RightToLeft بكفاءة ولا توجد حاجة لاستخدام تقنية المرآة عليها.

مدخلك إلى تقنية المرآة

تقنية المرآة Mirroring مصطلح يستخدم لوصف اتجاه المخرجات إلى الاتجاه العربي (من اليمين إلى اليسار Right to Left Layout)، كلمة المخرجات في هذا السياق كلمة عامة بحيث تشمل اي شيء تراه عينك على الشاشة (صور، أدوات، نصوص، ... الخ). ظهرت التقنية مع بدايات نظم التشغيل Windows 98، ولكنها كانت مدعومة في النسخ العربية منها Enabled و Local فقط، اما مع الاصدارات Windows 2000 وما بعده، فتقنية المرآة أصبحت مدعومة في كافة النسخ والإصدارات.

إن أخذت مرآة ووضعيتها بجانب يدك، ستلاحظ ان صورة يدك في المرآة قد انعكست، سبب الانعكاس هو ان الإحداثي السيني قد تم عكسه فقط، كذلك الحال مع تقنية المرآة، فكل ما تقوم به هذه التقنية هو عكس الإحداثي الأفقي (شكل 14-15).



شكل 14-15: عكس الإحداثي السيني عن تطبيق تقنية المرآة.

لا تقتضي تقنية المرآة عكس الإحداثي السيني فقط، بل تتطلب عكس أفكارك ونظرتك إلى الأمور في كل شيء، فالإحداثي السيني سيزيد كلما اتجهنا يسارا ويقل كلما اتجهنا يمينا. الجملة

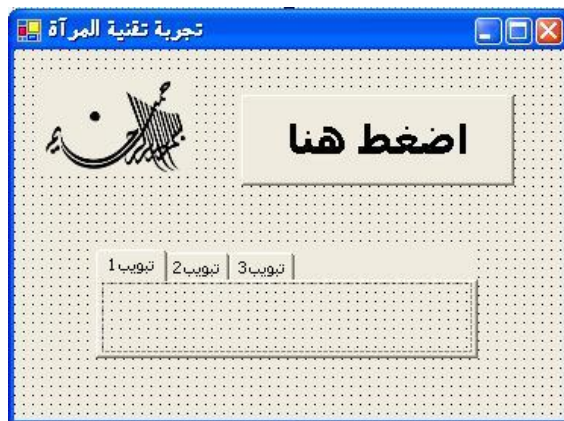
السابقة تشمل كل شيء يتعلق بالإحداثيات تراه امام عينك، الأدوات وخصائص الموقع، إحداثيات النقاط التي ترسلها الفأرة، الصور والرسوم باستخدام GDI+، وكل شيء اخر. اهم نقطة في هذا القسم من الفصل: **عملية العكس التي تنتجها تقنية المرآة تقع على الأداة والأدوات المحصورة بها فقط**. فمثلا، عند تطبيق تقنية المرآة على نافذة النموذج، وحاولت تحريك نافذة النموذج على سطح المكتب، استمر في التعامل معه كما تفعل سابقا، ولكن ان تعلقـت الشيفرات المصدرية داخل حدود النموذج، فالوضع سيبدأ بالانعكاس.

ملاحظة

عند تطبيق تقنية المرآة على النماذج والأدوات، فانسى كل شيء يتعلق بالخاصية RightToLeft، حيث استخدامك لهذه الخاصية سيعمي الأمور بدلا من تكحيلها!

تطبيق تقنية المرآة بـ Visual Basic .NET

دعنا الان نأخذ مثالا يطبق تقنية المرآة، أنشئ نافذة نموذج وضع بها ثلاث أدوات هي: PictureBox، Button، و TabControl (شكل 14-16).



شكل 14-16: نافذة نموذج بها أداة PictureBox، Button، و TabControl.

لتطبيق تقنية المرآة على نافذة النموذج، عليك ارسال القيمة WS_EX_LAYOUTRTL إلى **النمط الموسع Extended Style** إلى النافذة، يمكنك تغيير قيمة النمط الموسع عن طريق الإجراء SetWindowLong() وهو احد إجراءات API الشهيرة، وبما انني -للمرة الثانية- لست من المبرمجين الشجعان، فلن أتجرأ وانوي استخدام احد إجراءات API.

ملاحظة

النمط الموسع Extended Style تركيب خاص بنظام التشغيل Windows ونوافذه تحدد من خلاله بعض الصفات الظاهرية للنافذة كحدودها، الأزرار الظاهرة عليها، نمط النافذة... الخ. (راجع مكتبة MSDN والخاصة بمراجع إجراءات API لمزيد من التفاصيل حول النمط الموسع والإجراء SetWindowLong()).

سؤال اخر هام جداً، أين ومتى سنغير قيم النمط الموسع؟ والاجابة ستكون لحظة انشاء النافذة من قبل نظام التشغيل وليس من قبل **Windows Forms**، وفي الحقيقة Windows Forms لا تنشئ النوافذ من نفسها وانما تستخدم إجراء CreateWindow() (من إجراءات API) في بنيتها التحتية لانشاء النافذة، وحتى تتمكن من الوصول إلى لحظة انشاء النافذة من قبل نظام التشغيل، سنقوم باعادة قيادة Overrides الخاصة CreateParams. الخاصية CreateParams هي كائن من النوع CreateParams، يحتوي على الخاصية ExStyle التي يمكنك من تغيير النمط الموسع للنافذة، الشيفرة التالية ستطبق تقنية المرآة على النافذة (الشكل 14-17):

```
Protected Overrides ReadOnly Property CreateParams() As _
    System.Windows.Forms.CreateParams

    Get
        Const WS_EX_LAYOUTRTL As Integer = &H400000

        Dim MirrorExStyle As System.Windows.Forms.CreateParams
        MirrorExStyle = MyBase.CreateParams

        ' تطبيق تقنية المرآة '
        MirrorExStyle.ExStyle = MirrorExStyle.ExStyle Or
        WS_EX_LAYOUTRTL

        Return MirrorExStyle
    End Get
End Property
```



شكل 14-17: تم عكس كل شيء حتى النص في الأداة Button والصورة في الأداة PictureBox.

تلاحظ في (شكل 14-17)، ان تغيير النمط الموسع على نافذة النموذج لم يكتفي فقط بتغيير وعكس مواقع الأدوات المحصورة بها، بل قام بتغيير النمط الموسع على الأدوات المحصورة نفسها، مما أدى إلى عكس النص الموجودة في الأداة Button وعكس الصورة الموجودة في الأداة PictureBox. لذلك، عليك ارسال القيمة WS_EX_NOINHERITLAYOUT أيضا إلى النمط الموسع حتى نطلب من نظام التشغيل عدم تغيير النمط الموسع للأدوات المحصورة. صرح النقص الموجود في الشيفرة السابقة واضف بيانات القيمة :WS_EX_NOINHERITLAYOUT

```

Const WS_EX_NOINHERITLAYOUT As Integer = &H100000

MirrorExStyle.ExStyle = MirrorExStyle.ExStyle Or _
    WS_EX_LAYOUTRTL Or WS_EX_NOINHERITLAYOUT

```



شكل 14-18: تم عكس الأدوات بالشكل الصحيح دون تغيير أنماطها الموسعة.

ملاحظة

لوعدنا إلى (الشكل 14-17) ستلاحظ ان الأداة TabControl قد عكست بالشكل الصحيح ولم تتأثر النصوص التي عليها، السبب في ذلك يتعلق بأسلوب الرسم الذي يتبعه سياق الجهاز Context Device والخاص بالأداة TabControl، فمعظم الأدوات الأخرى (Button، Label، RadioButton، CheckBox... الخ) تتبع أسلوب رسم يعرف بـ GM_ADVANCED (راجع مكتبة MSDN لمزيد من التفاصيل)، أما الأدوات المعقدة الأخرى كـ ListView، TreeView، ToolBar لا تتبع هذا الأسلوب، لذلك لن تتأثر مخرجاتها النصية Textual.

متى ترسل القيمة WS_EX_NOINHERITLAYOUT:

لا يشترط دائما ارسال القيمة WS_EX_NOINHERITLAYOUT إلى النمط الموسع، حيث ان بعض الأدوات تحتوي على عناصر وأدوات أخرى بداخلها عليك تغيير نمطها الموسع حتى يتم تطبيق تقنية المرأة عليها بالشكل الصحيح، الجدول التالي يعرض لك الأدوات التي تتطلب استخدام القيمة WS_EX_NOINHERITLAYOUT (كما نقول مراجع MSDN):

الأداة	ضرورة استخدام
Form	WS_EX_NOINHERITLAYOUT
ListView	لا.
Panel	لا.
StatusBar	نعم.
TabControl	نعم.
TabPage	نعم.
ToolBar	لا.
TreeView	لا.

من الجدول السابق، يتضح لنا ان الأداة TreeView لا تتطلب منع العناصر المحصورة فيها من تغيير نمطها الموسع، لذلك لن نرسل القيمة WS_EX_NOINHERITLAYOUT لها. لعمل ذلك، اعد قيادة الخاصية CreateParams التابعة للأداة TreeView، وحتى تتمكن من فعل ذلك، عليك تعريف فئة جديدة مشتقة من الأداة TreeView:



```
Public Class ArabicTreeView
    Inherits System.Windows.Forms.TreeView

    Protected Overrides ReadOnly Property CreateParams() As _
        System.Windows.Forms.CreateParams

    Get
        Const WS_EX_LAYOUTRTL As Integer = &H400000
        Dim MirrorExStyle As System.Windows.Forms.CreateParams

        MirrorExStyle = MyBase.CreateParams

        MirrorExStyle.ExStyle = MirrorExStyle.ExStyle Or
        WS_EX_LAYOUTRTL

        Return MirrorExStyle
    End Get
End Property
End Class
```

مشاكل إضافية

توجد مشكلة بسيطة عند تطبيق تقنية المرآة تتمحور حول الصور المعروضة على الأداة، فعند تطبيق تقنية المرآة على الأدوات التي تعرض صور في عناصرها (كالأداة TreeView، ListView، و ToolBar)، سيتم عكس اتجاه الصور أيضا (شكل 14-19).



شكل 14-19: صور موجودة في الأداة ToolBar تم عكسها

يمكنك حل المشكلة السابقة بأسلوبين، الأول في عكس الصور والرموز عند تصميمها ببرامج تحرير ومعالجة الصور، يعيب هذا الأسلوب ان الصور ستكون معكوسة ان طبقتها على أدوات لا تستخدم تقنية المرآة، اما الأسلوب الثاني هو عكس الصور برمجيا في شيفراتك المصدرية باستخدام طرق وخصائص الرسم ان كانت الصورة ستظهر على أداة تستخدم تقنية المرآة.

مشكلة اخرى تظهر مع الأدوات التي لا يمكنك اشتقاقها وراثية (NotInheritable) (كالأداة ImageList)، حيث لن تتمكن من إعادة قيادة خاصيتها CreateParams لتغيير نمطها الموسع، ويبقى الحل الفاشل هو بتغيير النمط الموسع لنافذة النموذج دون ارسال القيمة WS_EX_NOINHERITLAYOUT مما يؤدي إلى عكس كافة الأدوات المحصورة فيها (شكل 14-17).

أدوات صناديق الحوار الشائعة

بالنسبة لأدوات صناديق الحوار الشائعة (OpenFileDialog، SaveFileDialog، PrintDialog، ColorDialog، PageSetupDialog، و FontDialog) فستظهر على نوع نسخة نظام التشغيل ولغة واجهة المحلية Local.

صناديق الرسائل

صناديق الرسائل التي تعرضها باستخدام الدالة `MsgBox`، فيمكن تغيير اتجاهها إلى الاتجاه العربي بإرسال القيمتين `MsgBoxRight` و `MsgBoxRtlReading` مع الوسيلة الثانية للدالة:

```
MsgBox("هل تريد حذف الملف؟", MsgBoxStyle.MsgBoxRight _
    Or MsgBoxStyle.MsgBoxRtlReading _
    Or MsgBoxStyle.YesNo Or MsgBoxStyle.Question _
    Or MsgBoxStyle.DefaultButton2, "حذف ملف")
```



شكل 14-20: صندوق رسالة Message Box بالاتجاه العربي.

بالنسبة للغة الأزرار (كـ Yes و No) تعتمد على لغة نظام التشغيل المحلية Local، يمكنك تعديلها باستخدام مجموعة من إجراءات API إبحث عنها في مكتبة MSDN. أخيراً، إن استخدمت الإجراء `MessageBox`، فأرسل القيمتين `RtlReading` و `RightAlign` للوسيلة الأخيرة:

```
MessageBox.Show("حذف ملف", "هل تريد حذف الملف؟", _
    MessageBoxButtons.YesNo, _
    MessageBoxIcon.Question, MessageBoxDefaultButton.Button2, _
    MessageBoxOptions.RightAlign Or MessageBoxOptions.RtlReading)
```

ملاحظة

الدالة `MessageBox` تعمل مثل عمل الدالة `MsgBox` ولكن بصيغة مختلفة.

حاولت في هذا الفصل والفصل السابق تعريفك بنماذج وأدوات Windows Forms. وكما اتضح لك في السطور السابقة، تحتوي النماذج والأدوات على مئات الطرق، الخصائص، والأحداث التي تحتاج إلى وقت طويل للتعرف عليها واستخدامها بالشكل الأمثل. ودعني أخبرك، بأنه ستأتي إصدارات أخرى من إطار عمل .NET Framework. قبل الاستفادة من معظم هذه الأعضاء لأغلب المبرمجين. الفصل التالي يعرفك على تقنية GDI+ والتي تمكنك من تصميم واجهات استخدام رسومية متقدمة.

مبادئ + GDI

إن الله جميل يحب الجمال، والجمال لا تكتشفه سوى لغة العيون، وإن كان المبرمجين يحكمون على جودة البرامج من شيفراتها المصدريّة، فإن المستخدمين (وهم هدفنا الأول) نظرتهم محصورة على واجهاتها -أي من خارجها.

الفصل الخامس عشر **GDI+** يتعلق باستخدام تقنية **GDI+** الموجه بشكل مباشر للتعامل مع الصور والرسوم والمخرجات النصية. قسمت هذا الفصل إلى ثلاث أقسام يمثل كل قسم زاوية ومنظور لاستيعاب واستخدام تقنية **GDI+**.

ملاحظة

لاختصار كتابة الاسماء الطويلة لفئات **GDI+**، سأفترض انك قمت باستيراد مجالات الاسماء التالية:

```
Imports System.Drawing
Imports System.Drawing.Drawing2D
Imports System.Drawing.Imaging
Imports System.Drawing.Text
```

الرسم المتقدم

إن كان الرسامون يعتمدون على الريشة في التعبير ما بي داخلهم، فإن مبرمجي **.NET** يستخدمون مجموعة من الفئات معظمها في مجال الاسماء **System.Drawing.Drawing2D** للتعبير عن خواطرم الجياشة. هذا القسم مخصص بعمليات الرسم.

الكائن Graphics

قبل ان تبدأ باستخدام الريشة احم احم اقصد فئات الرسم، عليك تحضير ورقة الرسم اقصد سياق الجهاز **Device Context**. سياق الجهاز هو تركيب خاص بنظام التشغيل يحمل كل شيء يتعلق بمنطقة الرسم التي تنوي الرسم فيها، حتى تحصل على سياق رسم لابد ان تمتلك كائن من الفئة Graphics، وللأسف الشديد لن تستطيع إنشاء هذا الكائن مباشرة باستخدام New حتى تحصل عليه، بل عليك الحصول عليها بطريقة من طريقتين: الاولى عن طريق وسيطة بعض احداث الرسم -كالحدث Paint التابع للاداة أو نافذة النموذج:

```
Private Sub Form1_Paint(ByVal sender As Object, _
    ByVal e As PaintEventArgs) Handles MyBase.Paint

    Dim gr As Graphics = e.Graphics
    ...
    ...
End Sub
```

والثانية باستخدام الطريقة CreateGraphics() التابعة لنافذة النموذج والادوات:

```
Dim MainForm As New frmMain
...
...
Dim gr As Graphics = MainForm.CreateGraphics()
```

مع ذلك، يوجد اختلاف كبير بين الطريقتين من ناحية تفريغ المصادر، حيث ان استخدام سياق رسم جاهز لا يتطلب منك تحريره يدويا من الذاكرة عند عدم الحاجة له، وذلك ان الاداة أو نافذة النموذج (التي أخذت منها سياق الرسم) ستقتل هذا السياق لحظة موتها، اما إنشائك لسياق رسم خاص باستدعاء الطريقة CreateGraphics() يحصر المسؤولية عليك كمبرمج من تحرير هذا السياق عند عدم الحاجة له:

```
gr.Dispose()
gr = Nothing
```

بعد إنشائك لسياق الرسم، فانك جاهز الآن لاستخدام هذا السياق وبدء الرسم الفعلي باستدعاء عدد كبير من الطرق التابعة للفئة Graphics نفسها أو بعض الفئات الاخرى -كما ستريك الفقرات التالية.

رسم الخطوط، المستطيلات، والدوائر

توفر الفئة Graphics مجموعة من الطرق التي تمكنك من رسم أشكال مبسطة (كالخطوط، المستطيلات، الدوائر، الأقواس وغيرها). معظم هذه الطرق تم اعادة تعريفها Overloads بأشكال عديدة، إلا ان الوسيطة الاولى تكون دائما نوع القلم وهو كائن من الفئة Pen (سأحدث عنه لاحقا في الفقرة الكائن Pen)، كما يمكنك تحديد احداثيات النقاط على شكل وسيطات أو ارسالها دفعة واحدة على شكل كائن من النوع Rectangle. استدعي الطريقة DrawLine()، الطريقة DrawRectangle()، أو الطريقة DrawEllipse() لرسم -على التوالي- الخطوط، المستطيلات، الدوائر (يمكن ان تكون دوائر بيضاوية ايضا):



```
Private Sub Form1_Paint(ByVal sender As Object, _
    ByVal e As PaintEventArgs) Handles MyBase.Paint

    Dim gr As Graphics = e.Graphics
    gr.DrawLine(Pens.Black, 0, 0, 200, 200)
    gr.DrawRectangle(Pens.Red, New Rectangle(0, 0, 200, 200))
    gr.DrawEllipse(Pens.Blue, 0, 0, 200, 200)
End Sub
```

انظر ايضا

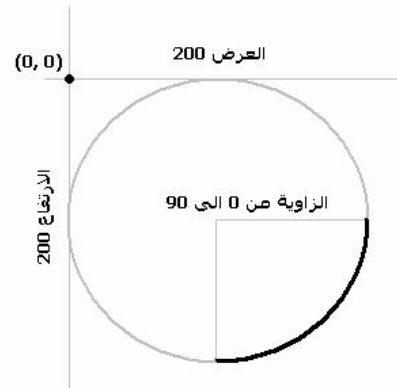
اذكر اني لمحت سابقا إلى الكائنات من النوع Rectangle في الفصل السابق **الأدوات Controls** عندما عرضت الخصائص المشتركة للموقع والحجم.

بالنسبة للأقواس فيمكنك رسمها باستدعاء الطريقة DrawArc() والتي تتطلب نفس وسيطات الطريقة DrawEllipse() السابقة بالإضافة إلى وسيطين تحدد فيهما زاوية البداية والنهاية للقوس بنفس اتجاه عقارب الساعة (وحدة القياس الدرجة وليس الراديان):



```
gr.DrawArc(Pens.Black, 0, 0, 200, 200, 0, 90)
```

(شكل 15-1 بأعلى الصفحة التالية) يوضح احداثيات الشيفرة السابقة.




شكل 15-1: أحداثيات الاستدعاء DrawArc (... , 0, 0, 200, 200, 0, 90).

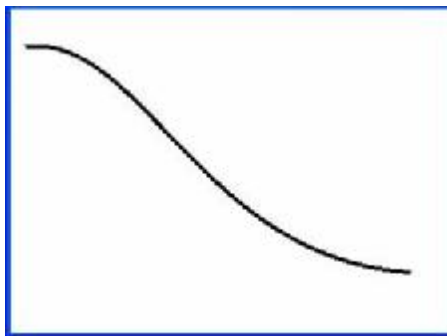
المزيد ايضا، لديك الطريقتين DrawLines() و DrawRectangles() والتي تستقبلان وسيطة مصفوفة من النوع Point و Rectangle على التوالي، هذا مثال لاستخدام الاولى:

```
Dim gr As Graphics = e.Graphics
Dim points() As Point = {New Point(10, 10), _
                          New Point(200, 200), New Point(10, 100), _
                          New Point(10, 10)}
gr.DrawLines(Pens.Green, points)
```

رسم المنحنيات المعقدة

يمكنك رسم منحنيات معقدة باستخدام الطريقتين DrawCurve() و DrawBezier()، راجع مكتبة MSDN للحصول على كافة التفاصيل حول وسيطات هذه الطرق ومدى تأثيرها، حيث سأكتفي هنا بعرض مثال لاستخدام الطريقة الثانية DrawBezier()، يبينه لك (الشكل 15-2):

```

Dim gr As Graphics = e.Graphics
gr.DrawBezier(Pens.Black, 10, 30, 100, 20, 140, 190, 300, 200)
```



شكل 15-2: منحنى معقد باستخدام الطريقة DrawBezier().

كائن القلم Pen

في الفقرات السابقة كنا نستخدم مجموعة معرفة من كائنات الاقلام في اطار عمل NET Framework بأسماء مثل: Pens.Black، Pens.Red، Pens.Green... الخ. يمكنك استخدام مجموعة اخرى من الاقلام تحمل ألوان النظام Colors في الفئة System Pens (مثل: SystemPens.ControlDark).

اما في هذه الفقرة فسنقوم بإنشاء اقلام خاصة بنا تحمل ألوان على أمزجتنا وسمكها اكثر من نقطة واحدة وأنماط خط مختلفة... الخ. نستطيع عمل ذلك بفضل الفئة Pen والتي يمكنك من إنشاء كائنات تمثل اقلام جديدة.

معظم مواصفات القلم يمكنك تحديدها في مشيد الفئة Pen، الشيفرة التالية ستنشئ قلمًا جديدًا لونه اسود وحجمه 4 بكسل:

```
Dim myPen As New Pen(Color.Black, 4)
```

يمكنك استخدام الكائن myPen مع اي طريقة من طرق الرسم التابعة للفئة Graphics السابقة، وذلك بارساله محل الوسيطة الاولى:

```
Dim gr As Graphics = e.Graphics
gr.DrawLine(myPen, 0, 0, 200, 200)
```

من الضروري التنبيه هنا بأنك انت المسئول الأول والأخير عن تحرير كائن القلم من الذاكرة ان لم تعد هناك حاجة لاستخدامه:

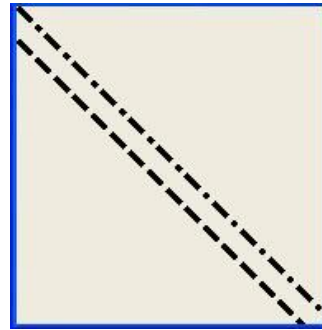
```
myPen.Dispose()
myPen = Nothing
```

المزيد ايضا، تستطيع تغيير نمط الخط (شكل 15-2) وذلك في ان تجعله منقطه
أو مقطع - - - - - أو مجموعة من الاشكال الاخرى تجدها في التركيب DashStyle والذي
تسنده إلى الخاصية التي تحمل نفس الاسم: DashStyle:

```
Dim gr As Graphics = e.Graphics
Dim myPen As New Pen(Color.Black, 4)

myPen.DashStyle = DashStyle.DashDot
gr.DrawLine(myPen, 0, 0, 200, 200)
myPen.DashStyle = DashStyle.Dash
gr.DrawLine(myPen, 0, 20, 200, 220)

myPen.Dispose()
```



شكل 15-2: تغيير نمط الخط DashStyle.

بل الأعظم من ذلك، يمكنك تخصيص شكل نمط الخط بنفسك عن طريق اسناد مصفوفة من النوع Single إلى الخاصية DashPattern، كل عدد من هذه المصفوفة يمثل عرض الشرطة - (ضع القيمة 1 ان اردت نقطة):

```
Dim gr As Graphics = e.Graphics
Dim myPen As New Pen(Color.Black, 7)
Dim customDash() As Single = {2, 1, 5, 1}
myPen.DashPattern = customDash
gr.DrawLine(myPen, 10, 10, 300, 10)
```

شيء ظريف جدا أعجبني في الكائن Pen -ومتأكد من انه سيعجبك انت ايضا- وهو قدرتك على تخصيص نقطة البداية والنهاية للخط عن طريق الخاصيتين StartCap و EndCap:



```
Dim gr As Graphics = e.Graphics
Dim myPen As New Pen(Color.Black, 7)
myPen.StartCap = LineCap.ArrowAnchor
myPen.EndCap = LineCap.ArrowAnchor
gr.DrawLine(myPen, 10, 40, 300, 40)
```

مخرجات الشيفرة السابقة وقبل السابقة يوضحها لك (الشكل 15-3):



شكل 15-3: تأثير الخصائص StartCap، EndCap، و DashPattern.

من ناحية اخرى، عند تعريف اقلام سمكها يزيد عن 1 بكسل، فينصح بشدة تحديد محاذاتها للاحداثيات، فكما تعلم ان الاحداثي للموقع (0, 0) يكون في نفس النقطة (0, 0) ان كان سمك الخط 1 بكسل، اما ان زاد السمك عن واحد بكسل فقد تكون النقطة فوق أو اسفل الاحداثي المحدد. لعمل ذلك، استخدم الخاصية Alignment لتحديد نوع المحاذاة:



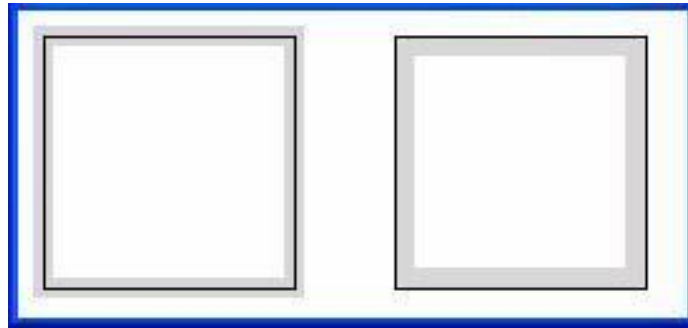
```
Dim gr As Graphics = e.Graphics
Dim myPen As New Pen(Color.LightGray, 8)

myPen.Alignment = PenAlignment.Center
gr.DrawRectangle(myPen, New Rectangle(10, 10, 100, 100))
gr.DrawRectangle(Pens.Black, New Rectangle(10, 10, 100, 100))

myPen.Alignment = PenAlignment.Inset
gr.DrawRectangle(myPen, New Rectangle(150, 10, 100, 100))
gr.DrawRectangle(Pens.Black, New Rectangle(150, 10, 100, 100))

myPen.Dispose()
```

الشيفرة السابقة ترسم اربعة مستطيلات بنفس الحجم توضح لك مدى تأثير الخاصية Alignment على مواقع رسم اقلامك الخاصة (شكل 15-4).



شكل 15-4: تأثير الخاصية Alignment على مواقع رسم الاقلام التي تزيد سمكها عن 1 بكسل.

كائن مسار الرسم GraphicsPath

كائن مسار الرسم ما هو إلا كائن يحتوي على جميع عمليات الرسم التي نودها، يمكنك اعتبار كائن مسار الرسم على أنه وعاء نضع فيه كافة طرق الرسم السابقة (بالصيغة AddLine()، AddRectangle()، AddCircle()... الخ) ومن ثم نرسمها دفعة واحدة.

الخطوة الاولى هي إنشاء كائن مسار رسم من الفئة GraphicsPath بالكلمة المحجوزة

:New

```
Dim myPath As New GraphicsPath()
```

دعنا نضيف ثلاث خطوط تمثل شكل مثلث:

```
myPath.AddLine(10, 10, 30, 60)
myPath.AddLine(30, 60, 60, 10)
myPath.AddLine(60, 10, 10, 10)
```

ان اردت اضافة المزيد من الخطوط، فعليك الانتباه بأنه سيتم اىصال النقطة الاخيرة من اخر خط رسمناها (10, 10) بالنقطة الاولى لاول خط سنرسمه، لذلك سنستدعي الطريقة StartFigure() لمنع حدوث ذلك:

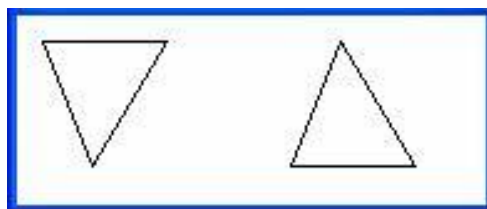
```
myPath.StartFigure()
myPath.AddLine(110, 60, 130, 10)
myPath.AddLine(130, 10, 160, 60)
```

لست بحاجة إلى إغلاق المثلث الأخير، حيث استدعاء الطريقة CloseFigure() يفى بالغرض:

```
myPath.CloseFigure()
```

والآن كل ما تحتاجه كائن سياق رسم Graphics واستدعاء الطريقة DrawPath() (شكل 16-6):

```
Dim gr As Graphics = e.Graphics
...
...
gr.DrawPath(Pens.Black, myPath)
myPath.Dispose()
```



شكل 16-6: مخرجات مسار الرسم GraphicsPath في الشيفرات السابقة.

ملاحظة

من الضروري قتل كائن مسار الرسم باستدعاء الطريقة Dispose() عند عدم الحاجة إليه.

التعبئة

تحتوي الفئة Graphics على ثمانية طرق معاد تعريفها Overloads بصيغ متعددة يمكنك من رسم أشكال معبئة Filled (يعني ملونة) هي: FillRectangle(), FillRectangles(), FillEllipse(), FillPie(), FillClosedCurve(), FillPath(), و FillRegion(). راجع مستندات .NET Documentation لمزيد من التفاصيل حول هذه الطرق.

الوسيلة الاولى لجميع هذه الطرق عبارة عن كائن من النوع Brush يمثل الفرشاة، يمكنك استخدام مجموعة جاهزة ومعرفة من الفرش أو تطوير فرش خاصة بك -كما سنفعل في الفقرة التالية كائن الفرشاة.

الفرق بين كائن الفرشاة وكائن القلم، هو ان كائن الفرش يستخدم لتحديد نقش ونمط التعبئة، بينما القلم خاص بحدود الخطوط والاشكال الاخرى التي رسمناها، يمكنك استخدام الفئة Brushes (أو استخدام SystemBrushes لالوان النظام System Color) لاستخدام انواع معرفة وجاهزة من فرش التعبئة:

```
Dim gr As Graphics = e.Graphics
' رسم دائرة سوداء '
gr.FillEllipse(Brushes.Black, 0, 0, 200, 200)
```

يمكنك استخدام الطريقة FillPath() لتلوين مسارات رسم (كائنات من النوع GraphicsPath) والتي تحدثنا عنها في الفقرة السابقة، اما الطريقة FillRegion() فتمكنك من تعبئة مناطق لكائنات من النوع Region (لم اتحدث عنها في هذا الكتاب).

كائن الفرشاة Brush

في الفقرة السابقة استخدمنا فرش معرفة وجاهزة للون التعبئة تابعة للفئات Brushes و SystemBrushes. وفي هذه الفقرة سنقوم بتطوير فرش خاصة بنا، وقبل ان نبدأ عليك معرفة ان جميع الفرش -سواء الجاهزة أو الخاصة بنا- مشتقة وراثيا من الفئة Brush. توجد مجموعة كبيرة من الفئات التي تمكنك من إنشاء فرش خاصة بك، وأسهلها الفئة SolidBrush التي تستقبل في مشيدها لون الفرشاة بحيث يكون هو نفسه لون التعبئة:

```
Dim gr As Graphics = e.Graphics
Dim myBrush As New SolidBrush(Color.Red)

gr.FillEllipse(myBrush, 0, 0, 200, 200)
myBrush.Dispose()
```

ملاحظة

لا تنسى قتل كائن الفرشاة باستدعاء الطريقة Dispose() عند عدم الحاجة إليه.

اما الفرش المنشئة من الفئة HatchBrush فانسب ما توصف به فئة النقش، حيث يمكنك من تحديد نقش من بين 56 نقش يدعمه التركيب HatchStyle (لا اعتقد انك ستطلب مني عرضها كلها، اليس كذلك؟!)، بالإضافة إلى تحديد لون الامامية والخلفية للنقش:

```
Dim gr As Graphics = e.Graphics
Dim myBrush As New HatchBrush(HatchStyle.BackwardDiagonal _
    , Color.Green, Color.White)


gr.FillRectangle(myBrush, 0, 0, 100, 100)

myBrush.Dispose()
```

ملاحظة

خاصية النقش HashStyle للقراءة فقط، ولا يمكن تعديلها على كائن الفرشاة الا بإنشاء نسخة كائن جديدة.

ولمن يبحث عن فنون التدرج اللوني، فيسرنى تعريفه بالفئة LinearGradientBrush والتي تمكنه من تحقيق ما يصبو اليه، ارسل حجم المقطع ولونا الامامية والخلفية واتجاه التدرج اللوني مع مشيد الفئة:

```

Dim gr As Graphics = e.Graphics
Dim myBrush As New LinearGradientBrush(New Rectangle(0, _
    0, 200, 200), Color.Black, Color.Blue, _
    LinearGradientMode.BackwardDiagonal)

gr.FillRectangle(myBrush, 0, 0, 300, 300)
myBrush.Dispose()
```

يمكنك تحديد الاختيار Tile عندما تتوي تغيير خلفية سطح مكتب خاصة ان كان حجم الصورة صغير، وذلك ليتم تكرار عرض الصورة حتى تغطي كافة سطح المكتب. هذا ما تفعله بالضبط الفرش من النوع TextureBrush، حيث تتطلب في مشيدها الصورة التي تود ان تمثل التعبئة في الاشكال المستخدمة لهذه الفرشاة:

```
Dim gr As Graphics = e.Graphics
Dim myBrush As New TextureBrush(PictureBox1.Image)

gr.FillRectangle(myBrush, 0, 0, 300, 300)
myBrush.Dispose()
```

توجد فئة أخرى متقدمة كثيرا لإنجاز فرش التعبئة، وموجهة للاستخدام بشكل خاص مع كائنات مسارات الرسم من النوع GraphicsPath، ابحث عنها في مكتبة MSDN تحت العنوان PathGradientBrush.

أنظمة القياس

في هذه الفقرة سأعرض عليك كيف يمكنك التحكم في أنظمة القياس Coordinate system والخاصة بكائن سياق الرسم Graphics لتتمكن من تحريك الاشكال من مواقعها ، تدوير العناصر وقلبها، وتحجيمها (تكبير/تصغير).

التحريك:

في الحقيقة، ان الذي سنقوم به هو عملية تغيير نظام القياس Coordinate system لكائن سياق الرسم Graphics، مما يعني تغيير موقع نقطة الاصل (0, 0)، فكما تعلم ان نقطة الاصل تكون في الزاوية العليا اليسرى من الاداة، مع ذلك قد تحتاج إلى تبسيطها في وسط الاداة لتسهيل عليك عملية رسم الدوال الرياضية.

لتغيير نظام القياس، استخدم الطريقة TranslateTransform() وارسل معها موقع نقطة الاصل:

```
Dim gr As Graphics = e.Graphics

' جعل نقطة الاصل وسط الاداة '
gr.TranslateTransform(Picture1.ClientRectangle.Width \ 2, _
    Picture1.ClientRectangle.Height \ 2)
```

عند استدعاء الطريقة TranslateTransform() مرة أخرى، ستتأثر بآخر تعديل لنقطة الاصل وقع على سياق الرسم، لذلك يفضل دائما استدعاء الطريقة ResetTransform() قبلها:

```
Dim gr As Graphics = e.Graphics

Gr.ResetTransform()
' جعل نقطة الاصل وسط الاداة '
gr.TranslateTransform(Picture1.ClientRectangle.Width \ 2, _
    Picture1.ClientRectangle.Height \ 2)
```

الشفيرة التالية تحاول رسم مربع بالطريقة DrawRectangle()، وبعد رسمه تغير نظام القياس وتنتقل موقع احداثيات نقطة الاصل إلى وسط النافذة بالطريقة TranslateTransform()، ومن ثم تحاول رسم المربع بنفس الاحداثيات المرسله قبل تعديل موقع نقطة الاصل (شكل 15-7):

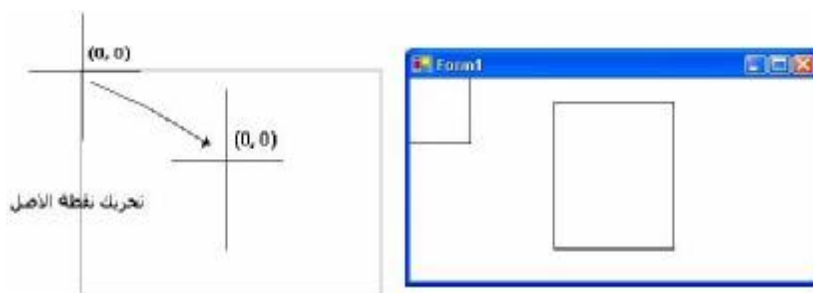


```
Private Sub Form1_Paint(ByVal sender As Object, ByVal e _
    As Forms.PaintEventArgs) Handles MyBase.Paint

    Dim gr As Graphics = e.Graphics

    gr.DrawRectangle(Pens.Black, -50, -50, 100, 100)
    gr.TranslateTransform(Me.ClientRectangle.Width \ 2, _
        Me.ClientRectangle.Height \ 2)

    gr.DrawRectangle(Pens.Black, -50, -50, 100, 100)
End Sub
```



شكل 15-7: تأثير الطريقة TranslateTransform() على كائن سياق الرسم.

الدوران:

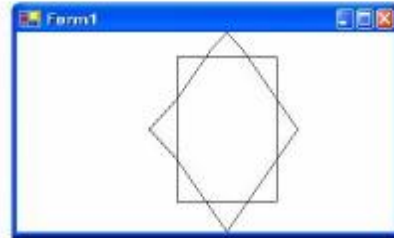
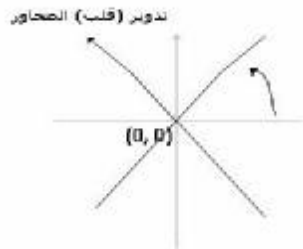
ان كانت الطريقة TranslateTransform() تغير موقع نقطة الاصل، فان الطريقة RotateTransform() تقلب المحور السيني والصادي x and y لنظام قياس الرسم والخاص بالكائن Graphics، ارسل مقدار زاوية القلب كوسيلة للطريقة لتتم تحريك المحاور باتجاه عقارب الساعة ان كانت القيمة موجبه، وعكس عقارب الساعة ان كانت القيمة سالبة. مخرجات الشيفرة التالية يظهرها لك (الشكل 15-8):



```
Dim gr As Graphics = e.Graphics

gr.TranslateTransform(Me.ClientRectangle.Width \ 2, _
    Me.ClientRectangle.Height \ 2)

gr.DrawRectangle(Pens.Black, -50, -50, 100, 100)
gr.RotateTransform(-45)
gr.DrawRectangle(Pens.Black, -50, -50, 100, 100)
```



شكل 15-8: تأثير الطريقة RotateTransform() على كائن سياق الرسم Graphics.

التحجيم:

يمكنك تحجيم نظام القياس بتكبيره أو تصغيره باستخدام الطريقة ScaleTransform() (شكل

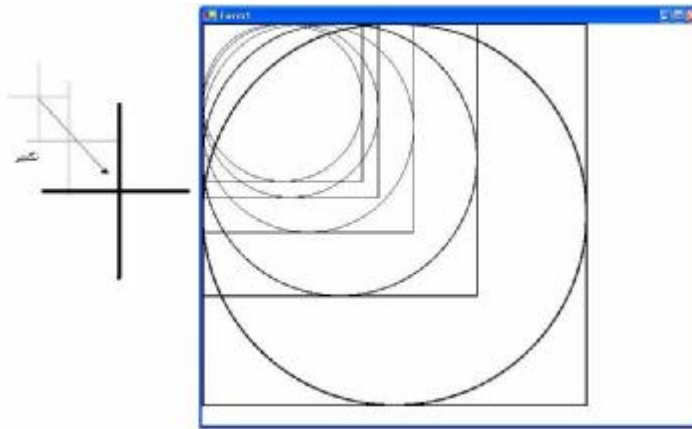
15-9):



```
Dim gr As Graphics = e.Graphics
Dim myPath As New GraphicsPath()
Dim counter As Single

myPath.AddRectangle(New Rectangle(0, 0, 200, 200))
myPath.AddEllipse(New Rectangle(0, 0, 200, 200))

For counter = 1 To 1.6 Step 0.1
    gr.DrawPath(Pens.Black, myPath)
    gr.ScaleTransform(counter, counter)
Next
```



شكل 15-9: تأثير الطريقة ScaleTransform() على كائن سياق الرسم Graphics.

وحدة القياس:

بشكل افتراضي، وحدة القياس لكائنات سياقات الرسم Graphics على الشاشة هي البكسل Pixel، يمكنك استخدام مجموعة من وحدات القياس الأخرى تسنّدها إلى الخاصية PageUnit هي: Inch، انش، Millimeter ملم، Point نقطة (75/1 انش)، Display عرض (72/1 انش)، أو Document مستند (300/1 انش) -تفيدك الوحدة الأخيرة كثيرا مع طابعات الليزر:

```
Gr.PageUnit = GraphicsUnit.Inch
```

التعامل مع الصور

المنظور الثاني الذي يمكننا من استيعاب تقنية GDI+ يتعلق بالصور والتعامل معها. في هذا القسم من الفصل سنحاول ان نلقي الضوء على مجموعة من الفئات في مجال الاسماء Imports System.Drawing.Imaging والتي تتعلق بالصورة الجاهزة.

تحميل وحفظ الصور

عند التعامل مع الصور، استخدم الفئة Image أو الفئة Bitmap، الأولى تحتوي على طرق وخصائص لفتح وحفظ الصور، والثانية مشتقة وراثيا من الأولى وتحتوي على مجموعة إضافية من الطرق والخصائص التي تتعامل مع محتويات الصورة نفسها.

يمكنك تحميل صورة من ملف باستدعاء الطريقة LoadFromFile()، أو من وحدة تخزين Stream باستدعاء الطريقة LoadFromStream() - كلا الطريقتين مدعومتان لكلا الفئتين:

```
Dim JPG As Image = Image.FromFile("C:\Ibrahim.JPG")
```

انظر ايضا

لمزيد من التفاصيل حول وحدات التخزين Streams، راجع الفصل الثامن الملفات والمجلدات.

اسلوب اخر أسهل لفتح الملف وذلك بارسال مساره إلى مشيد الفئة:

```
Dim JPG As New Bitmap("C:\Ibrahim.JPG")
```

ملاحظة

تدعم GDI+ مجموعة كبيرة من هياكل الصور PNG، BMP، GIF، JPEG، TIFF... الخ.

يمكنك في اي وقت من حفظ صورة في الكائنات النوع Bitmap أو Image باستدعاء طريقته Save() لحفظ ملف الصورة، تتطلب الطريقة اسم الملف والهئية المراد حفظها:

```
Dim JPG As New Bitmap("C:\Ibrahim.JPG")
JPG.Save("C:\Ibrahim.GIF", ImageFormat.Gif)
```

عند التعامل مع هياكل صور معقدة، ستتطلب الطريقة Save() معلومات اضافية كالععمق اللوني، نسبة الضغط... الخ.

عرض الصور

بمجرد حصولك على كائن من النوع Bitmap أو Image، يمكنك عرض صورته على اي سياق رسم باستدعاء الطريقة DrawImage() التابعة لسياق الرسم:



```
Private Sub Form1_Paint(ByVal sender As Object, _
    ByVal e As System.Windows.Forms.PaintEventArgs) _
    Handles MyBase.Paint

    Dim gr As Graphics = e.Graphics
    Dim JPG As New Bitmap("C:\Ibrahim.JPG")

    gr.DrawImage(JPG, 0, 0)
End Sub
```

ملاحظة

لا تحاول ابدا تحميل الملف من خلال الحدث Paint، وذلك بسبب كثر عدد مرات تنفيذ الحدث، مما يضعف كفاءة التنفيذ. ولكن كان غرضي في الأمثلة السابقة التوضيح فقط.

المزيد ايضا، الطريقة DrawImage() السابقة تم اعادة تعريفها Overloads باكثر من 30 صيغة، الصيغة السابقة تتطلب احداثي يمثل موقع رسم الصورة. صيغة اخرى مثل الصيغة السابقة، ولكن تضيف اليها المنطقة التي تود قطعها من الصورة الاصلية ورسمها (بارسال كائن من النوع Rectangle):

```
Dim gr As Graphics = e.Graphics
Dim JPG As New Bitmap("C:\Ibrahim.JPG")

gr.DrawImage(JPG, 0, 0, New Rectangle(10, 10, 50, 50),
    GraphicsUnit.Pixel)
```

صيغة ثالثة تمكنك من تحديد موقع وحجم المنطقة على سياق الرسم:

```
Dim gr As Graphics = e.Graphics
Dim JPG As New Bitmap("C:\Ibrahim.JPG")

gr.DrawImage(JPG, 0, 0, 200, 300)
```

كما يمكنك دمج الصيغتين الأخيرتين في خطوة واحدة، لتتمكن من قطع جزء من الرسمة و تحجيمها، الوسيطة الثانية تمثل المنطقة الهدف، والوسيطة الثالثة المنطقة المصدر (كلا الوسيطتين من النوع Rectangle):



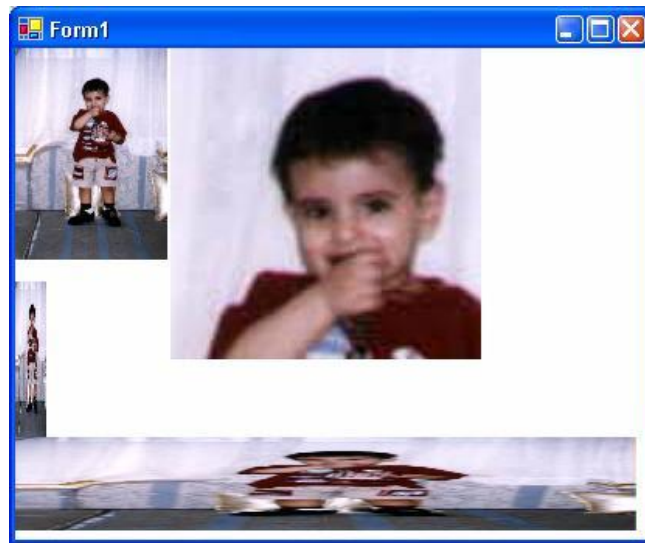
```
Dim gr As Graphics = e.Graphics
Dim destRect As New Rectangle(0, 0, 200, 200)
Dim sourceRect As New Rectangle(30, 30, 50, 50)
Dim JPG As New Bitmap("C:\Ibrahim.JPG")

gr.DrawImage(JPG, destRect, sourceRect, GraphicsUnit.Pixel)
```

ملاحظة

حتى تفرق بين الصيغ المختلفة، استعن بمحرر الشيفرة أو قراءة مستندات ..NET Documentation

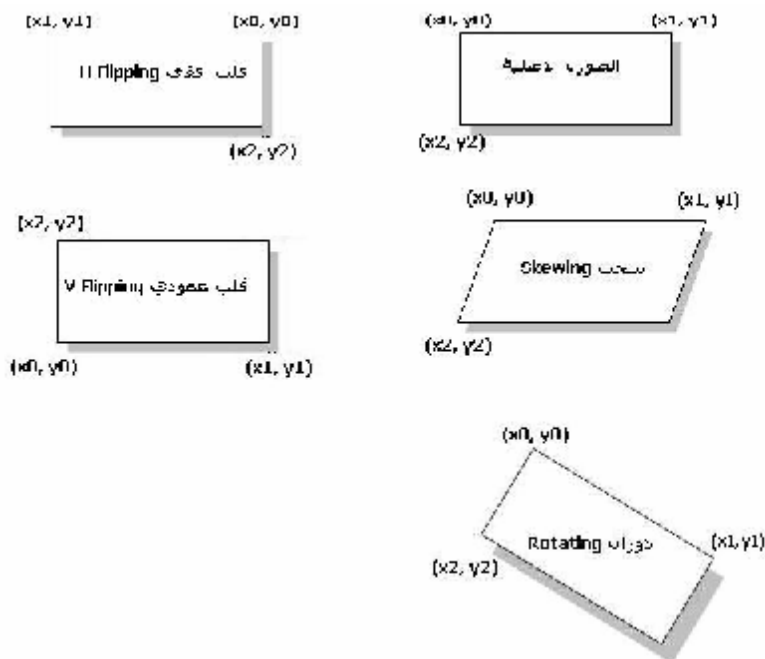
(شكل 10-15) يظهر لك مخرجات متعددة لصيغ الطريقة DrawImage().



شكل 10-15: مقاطع وأحجام مختلفة من الصورة.

عكس، قلب، وسحب الصور

من احد صيغ الطريقة DrawImage() المعاد تعريفها صيغة تقبل مصفوفة من النوع Point، حجم هذه المصفوف ثلاث عناصر، العنصر الاول (x0, y0) يمثل النقطة العلوية اليسرى، العنصر الثاني (x1, y1) يمثل النقطة العلوية اليمنى، اما العنصر الثالث (x2, y2) فيمثل السفلية اليسرى راقب المربع "الصورة الاصلية" في (الشكل 11-15).



شكل 11-15: تحديد الاحداثيات لقلب، سحب، وتدوير الصورة.

كل ما هو مطلوب منك الان اسناد القيم المناسب في المصفوفة Point وارسالها إلى الطريقة DrawImage()، لنبدأ مثلاً بالقلب الأفقي H Flipping والذي يتطلب عكس النقاط اليمنى باليسرى فقط (بافتراض ان احداثي نقطة البداية (x0, y0) للصورة الاصلية هي (0, 0):

```
Dim gr As Graphics = e.Graphics
Dim JPG As New Bitmap("C:\Eiffel.JPG")
Dim points() As Point = {New Point(JPG.Width, 0), _
    New Point(0, 0), New Point(JPG.Width, JPG.Height)}

gr.DrawImage(JPG, points)
```

اما القلب العمودي V Flipping، فيتطلب عكس النقاط العلوية بالسفلية (بافتراض ان احداثي نقطة البداية (x0, y0) للصورة الاصلية هي (0, 0) ايضا):



```
Dim gr As Graphics = e.Graphics
Dim JPG As New Bitmap("C:\Eiffel.JPG")
Dim points() As Point = {New Point(0, JPG.Height), _
    New Point(JPG.Width, JPG.Height), New Point(0, 0)}

gr.DrawImage(JPG, points)
```

بالنسبة للسحب Skewing، فيعتمد اعتماد كلي على مقدار السحب، وذلك وضعت المتغير L في الشيفرة التالية ليمكنك من زيادة/انقاص مسافة السحب (بافتراض ان احداثي نقطة البداية (x0, y0) للصورة الاصلية هي (0, 0) ايضا):



```
Dim gr As Graphics = e.Graphics
Dim JPG As New Bitmap("C:\Eiffel.JPG")
Dim L As Integer = 50
Dim points() As Point = {New Point(L, 0), New Point(JPG.Width + L,
    0), _
    New Point(0, JPG.Height)}

gr.DrawImage(JPG, points)
```

اخيرا، بالنسبة للقلب Rotating، فكان يمكنني وضع قيم ابتدائية للاحداثيات مباشرة، ولكن تعمدت إلى تعقيد الشيفرة اكثر وذلك للتسهيل عليك ووضع قيمة للزاوية بالدرجة في المتغير Angle بالشيفرة التالية (بافتراض ان احداثي نقطة البداية (x0, y0) للصورة الاصلية هي (0, 0) ايضا، لذلك اضطررت إلى استخدام قيمة سالبة في العنصر الثالث للمصفوفة (points):



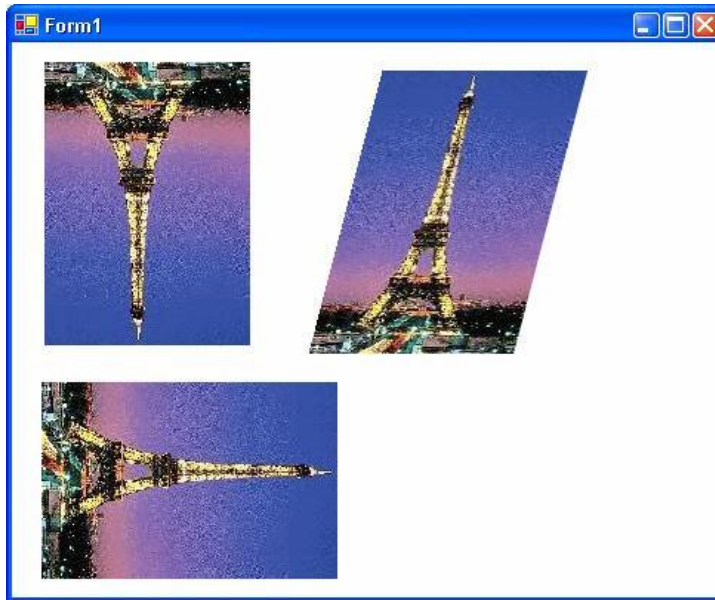
```
Dim gr As Graphics = e.Graphics
Dim JPG As New Bitmap("C:\Eiffel.JPG")
Dim Angle As Integer = 90
Dim AngleInRad As Single = CSng(Angle / (180 / Math.PI))
```

```
Dim a As Single = CSng(Math.Cos(AngleInRad))
Dim b As Single = CSng(Math.Sin(AngleInRad))
Dim points() As PointF = {New PointF(0, 0), _
    New PointF(JPG.Width * a, JPG.Width * b), _
    New PointF(-JPG.Height * b, JPG.Height * a)}
```

gr.DrawImage(JPG, points)

ملاحظة

الفئة PointF هي تماما مثل الفئة Point، ويمكن الفرق ان الاولى تحمل خاصيتين X و Y من النوع Single، بينما الثانية من النوع Integer.



شكل 15-12: قلب عمودي للصورة، سحبها بمقدار 50 بكسل، وتدويرها بمقدار 90 درجة.

تحديد الألوان

يمكنك تحديد اللون الخفي Transparent Color في الصورة بإرسال قيمة اللون كوسيلة مع الطريقة MakeTransparent() (شكل 13-15):



```
Dim gr As Graphics = e.Graphics
Dim JPG As New Bitmap("C:\dev4arabs.JPG")

gr.DrawImage(JPG, 0, 0)
JPG.MakeTransparent(Color.White)
gr.DrawImage(JPG, 0, 100)
```



شكل 13-15: اختيار اللون الأبيض كلون مخفي.

المزيد ايضا، تستطيع تحديد درجة إخفاء اللون المخفي من الصورة (كما تفعل مع الخاصية Opacity لنوافذ النماذج) وذلك بإرسال كائن من النوع ImageAttribute إلى الطريقة DrawImage() (التابعة كائن سياق الرسم Graphics)، ولو كان الأمر بهذه السهولة لعرضت عليك مثال، إذ يتطلب الكائن ImageAttribute مصفوفة ثنائية البعد Two Dimensional Array ترسلها كوسيلة إلى طريقته SetColorMatrix، لذلك انصحك بالرجوع إلى مستندات NET Documentation. للاستخدام الأمثل لها.

الرموز Icons

الفئة Icon تمثل رمز ايقونة من رموز نظام التشغيل Windows، يمكنك استعمال هذه الفئة لتحميل ورسم الرموز Icons، مع ذلك هذه الفئة ليست مشتقة من الفئة Image، فمعظم الطرق والخصائص التي تطرقت لها سابقا ليست مدعومة فيها.

يمكنك تحميل ملف الرمز بإرساله كمشيد للفئة Icon (تستطيع استخدام وحدة تخزين Stream أيضا):

```
Dim icon As New Icon ("C:\test.ico")
```

طريقة أخرى يمكنك من الحصول على رمز هي باستخدام كائن رموز تابعة للفئة SystemIcons، تعود بمجموعة من الرموز الجاهزة تعتبر شائعة بين تطبيقات Windows:

```
Dim icon As Icon = SystemIcons.WinLogo()
```

بعد حصولك على مرجع للرمز في الكائن Icon، تستطيع إرساله كوسيلة لكائن سياق الرسم Graphics وعرضه فوراً:

```
Dim gr As Graphics = e.Graphics
Dim icon As New Icon("C:\dev4arabs.ico")
gr.DrawImage(icon, 0, 0)
```

لا تنسى قتل كائن الرمز عند عدم الحاجة إليه باستدعاء الطريقة Dispose():

```
Icon.Dispose()
```

كما أخبرتك سابقاً، الفئات من النوع Icon محدودة الإمكانيات، فهي ليست مشتقة من الفئة Image، وإن اردت تطبيق الطرق والخصائص التي ذكرتها في الفقرات السابقة، يمكنك تحويل الرمز إلى صورة من النوع Bitmap باستخدام الطريقة ToBitmap():

```
Dim gr As Graphics = e.Graphics
Dim icon As New Icon("C:\dev4arabs.ico")
Dim BMP As Bitmap = icon.ToBitmap()

JPG.MakeTransparent(Color.White)
gr.DrawImage(BMP, 0, 100)

icon.Dispose()
BMP.Dispose()
```

اخيرا، كائن سياق الرسم Graphics يحتوي على الطريقة DrawIconUnstretched()، والتي تستقبل كائن رمز Icon، بحيث ترسل معها كائن منطقة من النوع Rectangle يتم تكرار عرض الرمز في هذه المنطقة.

المخرجات النصية

الجزء الثالث والآخر من GDI+ يتعلق بالمخرجات الحرفية النصية والتي تستخدم الخطوط Fonts بكافة اشكاله، لوانه، احجامه، وانماطه لعرض الحروف. ومعظم الفئات التي سنتعامل معها في هذا القسم مشمولة في مجال الاسماء الخاص بالمخرجات النصية System.Drawing.Text.

قبل البدء في عرض المخرجات النصية، وتنسيق خطوطها ومحاذاة حروفها، من الجيد التعرف على عوائل الخطوط واخذ فكرة عامة حولها، حيث ستحتاج فئاتها كثيرا ان رغبت في معرفة الخطوط وأنماطها المدعومة في الجهاز.

عوائل الخطوط

عائلة الخط Font Family هي مجموعة مترابطة من انماط مختلفة للخط ولكن بشكل واحد، فمثلا افراد عائلة الخط Tahoma الموقرة تتكون من: Tahoma Regular (العادي)، Tahoma Bold (سميك)، Tahoma Italic (مائل)، و Tahoma Bold Italic (سميك ومائل). يمكنك معرفة جميع عوائل الخطوط المثبتة في الجهاز عن طريق الخاصية Families والتابعة للكائن InstalledFontCollection والتي تعود بمصفوفة كائنات من النوع :FontFamily

```
Dim fonts As New InstalledFontCollection()
Dim fontFamilies() As FontFamily = fonts.Families

Dim fontFamily As FontFamily

For Each fontFamily In fontFamilies
    ...
    ...
    ...
Next
```

ملاحظة

الخاصية Families لا تعود إلا بعوائل الخطوط من النوع TrueType و OpenType فقط.

طريقة اخرى تمكّنك من معرفة عوائل الخطوط هي الطريقة المشتركة FontFamily.GetFamilies() والتي تتطلب كائن سياق جهاز من النوع Graphics:

```
Dim gr As Graphics = Me.CreateGraphics
Dim fontFamilies() As FontFamily = FontFamily.GetFamilies(gr)
gr.Dispose()
```

الفرق بين هذه الطريقة والخاصية Families السابقة، ان هذه الطريقة تعود بعوائل الخطوط التي يمكن عرضها واستخدامها على سياق الجهاز Graphics المرسل (فلا تنسى انه توجد عوائل خطوط موجه للاستخدام على الشاشة أو على الطابعة بشكل حصري). كما يمكنك إنشاء كائن من النوع FontFamily مباشرة بارسال قيمة حرفية تمثل اسم عائلة الخط مع مشيده:

```
Dim fontFamily As New FontFamily("Tahoma")
```

بمجرد حصولك على مرجع لعائلة الخط في الكائن FontFamily، يمكنك الاستعلام عن الانماط المختلفة للخط باستخدام الطريقة IsAvailable()، والتي ترسل معها النمط المطلوب لتعود بالقيمة True ان كان مدعوما في الجهاز الحالي:

```
Dim fontFamily As New FontFamily("Tahoma")
If fontFamily.IsStyleAvailable(FontStyle.Bold) Then
    ...
End If
```

رسم النصوص

يقصد بعبارة رسم النصوص اي كتابة المخرجات الحرفية على سياقات الرسم Graphics، وان نظرته من جانب تقني، عملية كتابة النصوص ما هي الا رسم تلك لمجموعة من الاشكال تمثل الحروف. على العموم، تحتاج إلى إنشاء كائن من النوع Font حتى تتمكن من الكتابة، يحتوي مشيد الكائن على 13 صيغة معاد تعريفها Overloads، الاولى تتطلب اسم وحجم الخط:

```
Dim font1 As New Font("Tahoma", 12)
```

صيغة اخرى تضيف لها نمط Style الخط:

```
Dim font2 As New Font("Tahoma", 20, FontStyle.Bold)
Dim font3 As New Font("Tahoma", 20, FontStyle.Bold Or
FontStyle.Italic)
```

وصيغة ثالثة تمكنك من استخدام كائن عائلة خط FontFamily:

```
Dim fontFamily As New FontFamily("Arial")
Dim font4 As New Font(fontFamily, 20, FontStyle.Bold)
```

ملاحظة

خصائص الكائنات من النوع Font للقراءة فقط ReadOnly ولن تتمكن من تعديلها بعد إرسالها إلى المشيد. إن أردت تغيير قيمة احد الخصائص، أنشئ الكائن من جديد.

حجم الخط الذي تختاره تكون وحدته خاصة بنظام التشغيل (وحدة مترية وليس بكسلية)، مع ذلك تستطيع تغيير الوحدة بارسال تركيب Enum بالاسم GraphicsUnit:

```
Dim myFont As New Font("Tahoma", 20, FontStyle.Bold,
GraphicsUnit.Pixel)
```

بعد إنشائك لكائن الخط وتنسيق كافة خصائصه، تستطيع البدء باستخدام فورا على اي كائن سياق جهاز يدعم ذلك الخط، استدعي طريقة سياق الجهاز DrawString() لرسم النصوص، تتطلب وسيطتها الاولى النص String المراد رسمه، الثانية كائن الخط Font، الثالثة كائن الفرشة Brush، والبقية احداثيات موقع الرسم:



```
Dim gr As Graphics = e.Graphics
Dim myFont As New Font("Tahoma", 20, FontStyle.Bold)

gr.DrawString("شبكة المطورون العرب", myFont, Brushes.Black, 0, 0)
```

اخيراً، يمكنك الاستعلام عن المساحة المطلوبة لعرض النص بخط معين باستخدام الطريقة `MeasureString()` والتي تعود بكائن من النوع `SizeF` بعد ارسالك لوسيطتين الاولى تمثل النص والثاني كائن الخط `Font`:

```
Dim size As SizeF
Dim myText As String = "..."
Dim myFont As New Font (...)

size = gr.MeasureString(myText, myFont)
```

الصيغة السابقة للطريقة تفرض بان النص سيعرض في سطر واحد، يمكنك معرفة الارتفاع المطلوب ان حددت العرض `Width` عندما تتوي التفاف النص `Wrap`:

```
gr.MeasureString(myText, myFont, 200)
```

التفاف النص

الطريقة `DrawString()` يمكن ان تستقبل وسيطة من النوع `RectangleF` تحدد بها المنطقة التي تود من النص ان يظهر عليها، وبحيث يتم التفافه تلقائياً ان تجاوز حدودها:



```
Dim gr As Graphics = e.Graphics
Dim myFont As New Font("Tahoma", 14)
Dim text As String = "ان لم تجمعنا الايام جمعنا الذكريات"
"، واذا العين لم تراك فالقلب لن ينسك"

gr.DrawString(text, myFont, Brushes.Black, New _
    RectangleF(10, 10, 200, 200))
' ارسم حد اضافية
gr.DrawRectangle(Pens.Black, New Rectangle(10, 10, 200, 200))
```



شكل 15-14: حجر النص في منطقة من النوع RectangleF.

ملاحظة

الفئة RectangleF تماثل الفئة من النوع Rectangle، والفرق الوحيد بينهما ان الاولى تستخدم قيم من النوع Single لخصائصها، بينما الثانية من النوع Integer.

اعلم انك ستتساءل عن طريقة ما تمكنا من محاذاة النص (شكل 15-14) بحيث يكون في الاتجاه العربي من اليمين إلى اليسار Right-to-Left، والإجابة ستكون شافية وواقية عن طريق الكائن StringFormat والذي يستحق فقرة كاملة لقوته وجبروته.

الكائن StringFormat

يمكنك ارسال الكائن StringFormat إلى طريقة سياق الجهاز DrawString()، يحتوي هذا الكائن على مجموعة إضافية من الطرق والخصائص الموجه للتحكم في المخرجات الحرفية، لديك مثلاً الخاصية Alignment والتي تتحكم في محاذاة النص والتي تكون إما Near (اليسار)، Center (الوسط)، أو Far (اليمين):



```
Dim gr As Graphics = e.Graphics
Dim myFont As New Font("Tahoma", 14)
Dim sf As New StringFormat()
Dim text As String = " _ & "ان لم نجتمعنا الايام جمعتنا الذكريات"
    "، واذا العين لم تراك فالقلب لن ينساک"

sf.Alignment = StringAlignment.Far
gr.DrawString(text, myFont, Brushes.Black, _
    New RectangleF(10, 10, 200, 200) , sf)

' ارسم حد اضافية
gr.DrawRectangle(Pens.Black, New Rectangle(10, 10, 200, 200))
```

عند الحديث عن مخرجات الحروف العربية فلا تستخدم خاصية المحاذاة `Alignment`،
وانما اعتمد على الخاصية `FormatFlags` لتغيير اتجاه النص `Text Direction` بإرسال القيمة
`DirectionRightToLeft` لها (شكل 15-15 بالصفحة التالية):



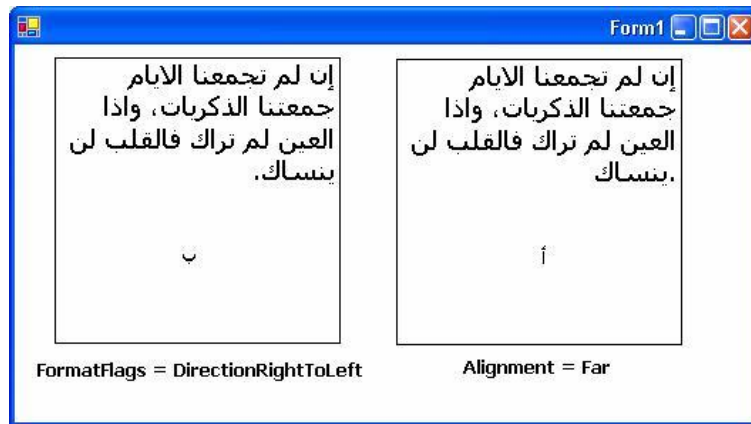
```
Dim gr As Graphics = e.Graphics
Dim myFont As New Font("Tahoma", 14)
Dim sf As New StringFormat()
Dim text As String = " _ & "ان لم نجتمعنا الايام جمعتنا الذكريات"
    "، واذا العين لم تراك فالقلب لن ينساک"

sf.FormatFlags = StringFormatFlags.DirectionRightToLeft
gr.DrawString(text, myFont, Brushes.Black, _
    New RectangleF(10, 10, 200, 200) , sf)

' ارسم حد اضافية
gr.DrawRectangle(Pens.Black, New Rectangle(10, 10, 200, 200))
```

ملاحظة

عند تغيير اتجاه النص عن طريق الخاصية `FormatFlags`، فإن اتجاه
النص في الخاصية `Alignment` سيصبح معاكس، أي أن القيمة
`Near` تحاذي النص إلى اليمين والقيمة `Far` إلى اليسار.



شكل 15-15: الفرق بين المحاذاة واتجاه النص.

قد لا تلاحظ الفرق بين تأثير المحاذاة واتجاه النص في (الشكل 15-15)، ولكن لو أعدت التدقيق ستلاحظ اختلاف موقع نقطة نهاية الجملة. ولكن كان هذا الفرق طفيفا، فصدقي ان الاختلاف اكبر بكثير من كونه تغيير موقع النقطة، جرب مثلا تعديل قيمة المتغير Text بحيث يحتوي على جملة تشمل كلمات إنجليزية وعربية، سيظهر الفرق عظيمًا هذه المرة (شكل 15-16).



شكل 15-16: الفرق بين المحاذاة واتجاه النص.

ملاحظة

ان كنت من مستخدمي Windows المتمرسين، فدعني أخبرك ان تغيير اتجاه النص هو تماما ما تحدّثه المفاتيح [Shift] + [Ctrl] اليمنى.

المزيد ايضا، يمكنك محاذاة النص بالنسبة للمنطقة Rectangle من منظور عامودي عوضا عن افقي عن طريق الخاصية LineAlignment التي إما تكون Near (فوق)، Center (وسط)، أو Far (اسفل).

المحاذاة الكلية Justify:

تعتمد حروف لغتنا الجميلة على الكشيدة للمحاذاة الكلية Justify للفقرة، حاولت البحث في مكتبة MSDN على اي وسيلة أو قيمة تمكننا من تطبيق الكشيدة للمحاذاة الكلية للفقرة، ولكن محاولتي - مع الاسف - لم تجد الا دعم لهذه المحاذاة مع ادوات Web Forms والخاصة بصفحات HTML فقط. لذلك، كان علي تطوير الفئة ArJustify التي تستخدم الكشيدة لمحاذاة النص (شكل 15-17).



شكل 15-17: المحاذاة الكلية باستخدام الكشيدة عن طريق الفئة ArJustify.

تتطلب الفئة ArJustify خمس وسيطات في مشيدها هي: سياق الرسم أو الجهاز، النص المراد محاذاته، كائن الخط Font، عرض المنطقة، واسند القيمة True ان اردت محاذاة نص السطر الاخير. تحتوي الفئة على الخاصية JustifiedText والتي تعود بالنص بعد إضافة الكشيدات له:

```
Dim myText As String

myText = New ArJustify(gr, myText, myFont, 200, True).JustifiedText

gr.DrawString(myText, myFont, Brushes.Black, _
    New RectangleF(250, 10, 200, 200), sf)
```

ان نظرنا في داخل شيفرة الفئة، فسنجد أهم اجرائين بها هما JustifiedText() و AddKashidas()، الاول يقوم بتوزيع الكلمات وفصلها لمعرفة أقصى عدد من الكلمات يمكن للسطر ان يحتويه، والثاني يقوم بالاضافة الفعلية للكشيدة:



```
Class ArJustify
...
...
...
Public ReadOnly Property JustifiedText() As String
    Get
        Return Me.text
    End Get
End Property

Private Sub JusitfyLines()
    Dim counter As Integer = 0

    Me.textLines = Me.text.Split(" "c)

    Do While counter <= UBound(Me.textLines)
        If gr.MeasureString(Me.textLines(counter), Me.font).Width
            = Me.width Then
            ' لا تفعل شي

        ElseIf gr.MeasureString(Me.textLines(counter), _
            Me.font).Width > Me.width Then

            ' عليك فصل الكلمة هنا
        Else
            ' ادمج الكلمات
            counter += 1
            Do While counter <= UBound(Me.textLines)
                If gr.MeasureString(Me.textLines(counter - 1) & _
```

```

        " "c & Me.textLines(counter), _
        Me.font).Width = Me.width Then

        Me.textLines(counter - 1) &= " "c & _
        Me.textLines(counter)
        RemoveWord(counter)
        counter -= 1
        Exit Do
    ElseIf gr.MeasureString(Me.textLines(counter - 1)_
        & " "c & Me.textLines(counter), Me.font).Width
        > Me.width Then

        counter -= 1
        Me.AddKashidas(counter)
        Exit Do
    Else
        Me.textLines(counter - 1) &= " "c & _
        Me.textLines(counter)
        RemoveWord(counter)
    End If
Loop
If counter >= UBound(Me.textLines) Then
    Me.AddKashidas(counter - 1)
End If
End If
counter += 1
Loop
Me.text = Me.text.Join(" "c, Me.textLines)
End Sub

Private Sub AddKashidas(ByVal index As Integer)
    Dim counter As Integer
    Dim canAddKashida As Boolean
    Dim allowedAfterLetters As String = "ضمئففغغعججسببئئمكظظ"
    Dim allowedBeforeLetters As String = _
        "ضمئففغغعججسببلا أنمكظؤؤرىةوظ"

    If Me.JustifyLastLine = False AndAlso index = _
        UBound(Me.textLines) Then
        Exit Sub
    End If

    If Me.textLines(index).Length <= 1 Then
        Exit Sub
    Else
        counter = Me.textLines(index).Length - 2
    End If

    Do While gr.MeasureString(Me.textLines(index), Me.font).Width

```

```

< Me.width

If allowedAfterLetters.IndexOf(Me.textLines(index). _
    Chars(counter)) <> -1 AndAlso _
    allowedBeforeLetters.IndexOf(Me.textLines(index). _
    Chars(counter + 1)) <> -1 Then

    canAddKashida = True
    If gr.MeasureString(Me.textLines(index).Insert( _
        counter + 1, "_"c), Me.font).Width > Me.width Then

        Exit Do
    Else
        Me.textLines(index) = Me.textLines(index).Insert(
-
            counter + 1, "_"c)
        Dim counter2 As Integer
        For counter2 = counter To 0 Step -1
            If Me.textLines(index).Chars(counter2) = " "c
Then
                counter = counter2 - 1
                Exit For
            End If
        Next
        If counter2 <= 0 Then counter = _
            Me.textLines(index).Length - 2
        End If
    Else
        counter -= 1
        If counter <= 0 Then
            If canAddKashida Then
                counter = Me.textLines(index).Length - 2
            Else
                Exit Do
            End If
        End If
    End If
End If
Loop
End Sub
End Class

```

كما تلاحظ في الشيفرة السابقة، تعتمد الفئة ArJustify على المصفوفات من النوع String وهذا هو احد اكبر أسباب بطئها، مع ذلك يمكنك الاعتماد على الفئة StringBuilder لزيادة السرعة، كما يمكن لك تغيير الخوارزميات المتبعة بها.

انظر ايضا

قد تحتاج إلى العودة إلى الفصل السادس الفئات الأساسية ان اردت معرفة الغرض من الفئة `StringBuilder`.

المزيد ايضا، يمكنك الاستفادة من الفئة `ArJustify` لتنسيق محاذاة حروف وكلمات الشعر العربي (الشكل 15-18).



شكل 15-18: الاستفادة من الفئة `ArJustify` لتنسيق الشعر العربي.

ملاحظة

كان غرضي من الفئة `ArJustify` توضيح فكرة تطبيق المحاذاة الكلية فقط، فلا تعتمدها في مشاريعك لاني لم اجري اي اختبارات إضافية عليها ولم اصل إلى الدقة المطلوبة التي توفرها معظم برامج معالجة النصوص.

الارقام الهندية:

لا استخدم الارقام الهندية في حياتي اليومية وذلك لعدم قناعتني بها، مع ذلك تتبع هذه الارقام الاعدادات الإقليمية لمجموعة كبيرة من الدول العربية، وقد يفضلها العديد من مستخدمين نظم التشغيل Windows.

بشكل مبني، الارقام ستظهر بالاعتماد على اعدادات البيئة الحالية في جهازك الشخصي، مع ذلك يمكن تعديلها عن باستدعاء الطريقة SetDigitSubstitution() التي تتطلب منك وسيطتين الاولى معرفة LCID للدولة والثانية ونظام الاعداد، الشيفرة التالية تستخدم الاعدادات الإقليمية العربية (المملكة العربية السعودية)، وترى مخرجاتها في (الشكل 15-19):

```

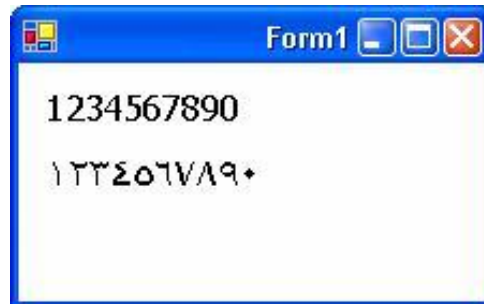
Dim gr As Graphics = e.Graphics
Dim sf As New StringFormat()
Dim myFont As New Font("Tahoma", 13)
Dim myText As String = "1234567890"

gr.DrawString(myText, myFont, Brushes.Black, _
    New RectangleF(10, 10, 280, 200), sf)

sf.SetDigitSubstitution(&H401, StringDigitSubstitute.Traditional)

gr.DrawString(myText, myFont, Brushes.Black, _
    New RectangleF(10, 40, 280, 200), sf)

```



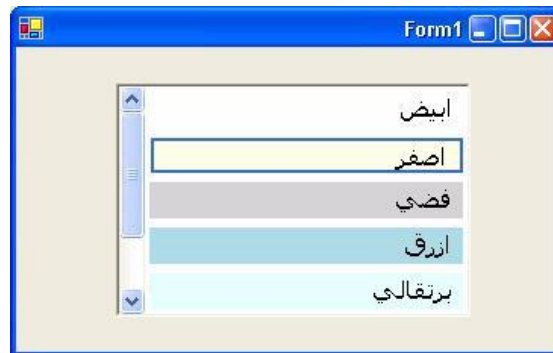
شكل 15-19: الارقام الهندية والعربية مدعومة في اعدادات العربية (المملكة العربية السعودية).

انظر ايضا

للحصول على معرفات LCID لباقي الدول العربية، راجع الفصل السادس الفئات الأساسية وبالتحديد الجدول الموجود صفحة 245.

عودة إلى الأدوات Controls

قبل ان اختتم هذا الفصل بودي عرض هذا القسم والتي ستفتح لك آفاق واسعة لتحسين التصميم الخارجي والهيكلية الظاهرية للأدوات دون الحاجة لإعادة بنائها من جديد. يمكنك بعض الادوات (كالاداة ListBox والقائمة MenuBox) من الوصول إلى سياق الجهاز Device Context الخاص بها، ماذا يعني هذا؟ يفيدك كرم الاداة بإعطائك مرجع لسياق رسمها من اضافة بعض اللمسات الفنية لمخرجات الاداة نفسها، مثلاً هب انك تريد اضافة عناصر لاداة ListBox تمكن المستخدم من اختيار لون معين، يستحسن في هذه الحالة ان يظهر اللون كخلفية لكل عنصر من عناصرها (شكل 15-20).



شكل 15-20: الاستفادة من سياق رسم الاداة لتغيير مظهر عناصرها.

حتى اريك مثلاً يستفيد من سياق جهاز الاداة، اصف اداة من النوع ListBox واكتب هذه الشيفرة في حدث Load الخاص بنافذة النموذج:



```
Public Class Form1
    Inherits System.Windows.Forms.Form
    ...
    ...

    Structure ColorName
        Dim ArabicName As String
        Dim Brush As Brush
    End Structure
    Dim colorItems(6) As ColorName

    Private Sub Form1_Load(ByVal sender As System.Object, _
        ByVal e As System.EventArgs) Handles MyBase.Load

        Dim ci As ColorName

        colorItems(0).ArabicName = "ابيض"
        colorItems(0).Brush = Brushes.White
        colorItems(1).ArabicName = "اصفر"
        colorItems(1).Brush = Brushes.Yellow
        colorItems(2).ArabicName = "فضي"
        colorItems(2).Brush = Brushes.Silver
        colorItems(3).ArabicName = "ازرق"
        colorItems(3).Brush = Brushes.Blue
        colorItems(4).ArabicName = "برتقالي"
        colorItems(4).Brush = Brushes.Orange
        colorItems(5).ArabicName = "احمر"
        colorItems(5).Brush = Brushes.Red
        colorItems(6).ArabicName = "اخضر"
        colorItems(6).Brush = Brushes.Green

        For Each ci In colorItems
            ListBox1.Items.Add(ci.ArabicName)
        Next

    End Sub
End Class
```

بالنسبة للاداة `ListBox`، فهي تحتوي على الخاصية `DrawMode` التي تسند لها القيمة `OwnerDrawFixed` لتسمح لك برسم عناصرها بنفسك، وبعد ذلك عليك اسناد قيمة مناسبة للخاصية `ItemHeight` لتحديد ارتفاع العنصر بالبكسل.

ستضع شيفرات الرسم بين فكي الحدث `DrawItem` والذي يتم تنفيذه بمجرد طلب الاداة منك رسم عنصر من عناصرها، يرسل هذا الحدث مع وسيطته مجموعة من الخصائص كالخاصية `Bounds` التي تحدد المنطقة المراد رسمها، الخاصية `Index` التي تمثل رقم العنصر المراد رسمه، والخاصية `State` التي تخبرك حالة العنصر (كان يكون محدد، ممكن، غير ممكن... الخ)،

اضف الشيفرة التالية بين فكي حدث الاداة DrawItem لتصمم اداة ملونة العناصر كالموجودة في
(الشكل 15-20):



```
Private Sub ListBox1_DrawItem(ByVal sender As Object, _
    ByVal e As System.Windows.Forms.DrawItemEventArgs) _
    Handles ListBox1.DrawItem

    Dim gr As Graphics = e.Graphics
    Dim myFont As New Font("Tahoma", 13)
    Dim rect As Rectangle = e.Bounds
    rect.Inflate(-3, -3)

    gr.FillRectangle(colorItems(e.Index).Brush, rect)

    If CBool(e.State And DrawItemState.Selected) Then
        rect.Inflate(-1, -1)
        gr.DrawRectangle(SystemPens.Highlight, rect)
        rect.Inflate(-1, -1)
        gr.DrawRectangle(SystemPens.Highlight, rect)
    End If

    gr.DrawString(colorItems(e.Index).ArabicName, _
        myFont, Brushes.Black, _
        rect.Width - gr.MeasureString(colorItems(e.Index).ArabicName, _
        myFont).Width, rect.Y)

    myFont.Dispose()
End Sub
```

مكتبة GDI+ غنية جدا وتحتوي على كثير من الفئات التي لم أتطرق لك، إن أردت التخصص في مجال الصور والرسوم بمكتبة GDI+، فلديك مستندات NET Documentation. لتبحر بين صفحاتها. والآن يمكنك الاستفادة من كل ما تعلمته في الفصول الثلاث السابقة، لتطور مشاريع من النوع Custom Controls أو Windows Services بالفصل التالي.

مواضيع متقدمة

عندما نتقدم في عمرك البرمجي مع Windows Forms، قد تحتاج إلى تطوير نوعية إضافية من المشاريع كالأدوات الخاصة Custom Controls أو خدمات Windows Services، والتي يمكنك إنجازها بكافة الأساليب المتبعة في الفصول السابقة.

في هذا الفصل اختتم معك الجزء الثالث **تطوير تطبيقات Windows** بالتحدث عن موضوع تطوير الأدوات الخاصة وخدمات Windows، كما سأختتم الفصل بعرض سريع لمجموعة من الفئات الإضافية والتي قد تحتاجها يوما من الأيام لبرامجك الموجه للعمل تحت Windows.

تطوير أدوات خاصة

تأتي مع رزمة Visual Studio .NET مجموعة كبيرة من الأدوات جاهزة الاستخدام، وقد تطرقنا إلى معظمها في الفصل الرابع عشر **الأدوات Controls**، مع ذلك قد تحتاج إلى تطوير المزيد من الأدوات الخاصة Custom Controls والتي تستخدمها في مشاريعك الخاصة، أو مشاريع أخرى.

قبل الاستمرار في هذا القسم، علينا ان نتفق على ثلاث مصطلحات هي: **المؤلف Author** وهو الشخص الذي يقوم ببرمجة الأداة، **المبرمج Programmer** وهو الشخص الذي يستخدم هذه الأداة في مشاريعه، **المستخدم User** وهو المستخدم النهائي للبرنامج الذي يعرض الأداة.

المزيد ايضا، **مرحلة التأليف Authoring Time** هي مرحلة تصميم وبرمجة الأداة، **وقت التصميم Design Time** هو وقت التصميم الخاص بالمبرمج Programmer لتصميم برنامجه واستخدام الأداة، **اما وقت التنفيذ Run Time** فهو الوقت تنفيذ البرنامج النهائي والذي يستخدم الأداة.

ثلاثة أساليب يمكنك اتباعها لتطوير أدوات خاصة أسهلها وراثشة أداة سابقة، استخدام مجموعة من الأدوات الجاهزة، أو إنشاء أداة مستقلة من الصفر. في الفقرات التالية اعرض لك هذه الأساليب الثلاث:

وراثه أداة

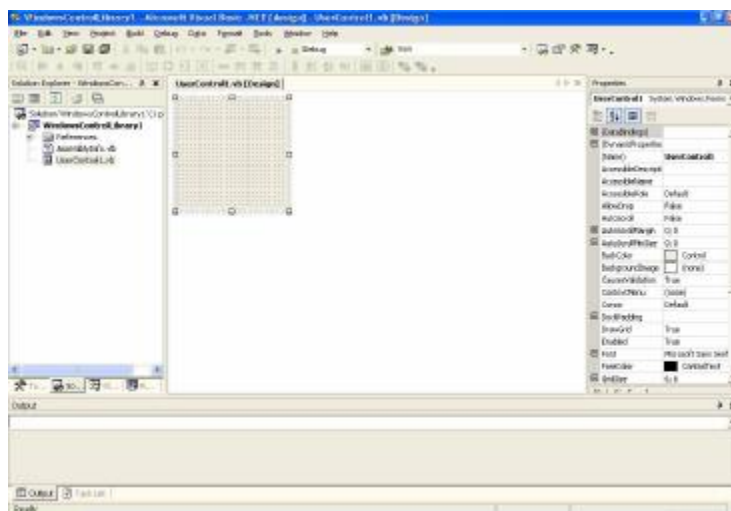
اسهل اسلوب تتبعه لانشاء أداة خاصة هو بوراثه أداة معينة لتطويرها وزيادة خصائصها وطرقها حتى ثلاثم احتياجائك، واذكر أني في الفصل الرابع عشر الأدوات **Controls** اضطرت لوراثه مجموعة من الأدوات (كـ TreeView) حتى أتمكن من تطبيق تقنية المرآة Mirroring عليها بالشكل الصحيح.

يعيب هذا الاسلوب ان الأداة لابد من ان تكون قابلة للاستئاق الوراثي حتى تتمكن من إتباعه، فبعض الأدوات (كالأداة ProgressBar أو ImageList)، لا يمكنك استئاقها وراثيا وبذلك لن تتمكن من اتباع هذا الاسلوب.

مع ذلك، اغلب الأدوات -المستخدمة بكثرة- قابلة للوراثه (كالأدوات Label، TextBox، ListView، Picture وغيرها) وبذلك يمكنك زيادة خصائصها وطرقها بحيث تناسب احتياجاتك الخاصة.

تأليف الأداة:

الخطوة الاولى لانشاء الأداة هو انشاء مشروع لها، اختر الامر New->Project من قائمة File وحدد الرمز Windows Control Library، ستتشئ لك بيئة التطوير Visual Studio .NET مشروع جديد من هذا النوع يحتوي على الملف UserControl1.vb (شكل 1-16). بالنسبة للنافذة المفتوحة امامك، فهي تمثل الأداة التي تود تصميمها والتعامل معها سيكون كما نتعامل مع نوافذ النماذج.



شكل 16-1: مشروع جديد من النوع Windows Control Library.

ملاحظة

المشاريع من النوع Windows Control Library ما هي إلا مشاريع من النوع Class Library تقليدية، ولا يوجد فرق جوهري بينها، فكل ما في الامر هو اختلاف الشيفرة البرمجية التي تولدها بيئة التطوير Visual Studio .NET بشكل مبدئي، واعدادات مختلفة في صندوق حوار خصائص المشروع Project Property Pages.

غير اسم المشروع إلى الاسم TextBoxExProject وافتح نافذة محرر الشيفرة التابعة للأداة UserControl1 وامسح كل شيء فيها، واكتب هذه الفئة التي تشتق الأداة TextBox:

```
Public Class TextBoxEx
    Inherits System.Windows.Forms.TextBox

    Sub New()
        MyBase.New()
    End Sub
End Class
```

بمجرد تعريفك للفئة TextBoxEx السابقة، فانك قد انتهيت من تطوير أداة TextBox كاملة، تحتوي على جميع خصائص، طرق، واحداث الأداة TextBox. مع ذلك، لا توجد أي فائدة من الأداة الجديدة ما لم تضيف أعضاء بها. سنضيف الخاصية AutoSelect والتي تسند لها القيمة True ان اردت تحديد كافة النص بمجرد حصول الأداة على تركيزها:

```
Private m_autoSelect As Boolean
Public Property AutoSelect() As Boolean
    Get
        Return m_autoSelect
    End Get
    Set(ByVal Value As Boolean)
        m_autoSelect = Value
    End Set
End Property
```

انسب حدث نضع في امر تحديد كافة النص في الأداة هو الحدث Enter، والذي سيتم تنفيذه بمجرد انتقال التركيز إلى الأداة:

```
Private Sub TextBoxEx_Enter(ByVal sender As Object, _
    ByVal e As System.EventArgs) Handles MyBase.Enter

    Me.SelectAll()
End Sub
```

وبذلك، تكون الشيفرة النهائية للأداة بهذا الشكل:



```
Public Class TextBoxEx
    Inherits System.Windows.Forms.TextBox

    Sub New()
        MyBase.New()
    End Sub

    Private m_autoSelect As Boolean
    Public Property AutoSelect() As Boolean
        Get
            Return m_autoSelect
        End Get
        Set(ByVal Value As Boolean)
            m_autoSelect = Value
        End Set
    End Property
```

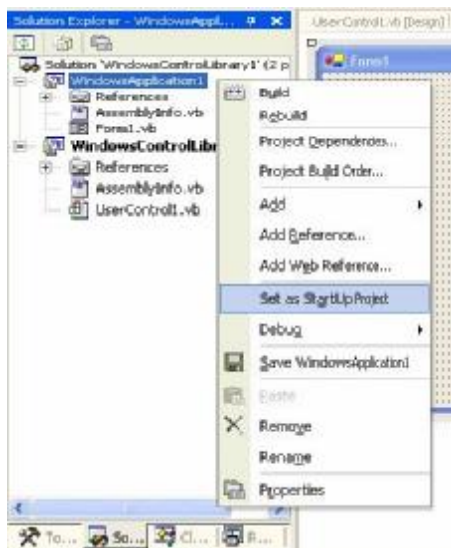
```
Private Sub TextBoxEx_Enter(ByVal sender As Object, _
    ByVal e As System.EventArgs) Handles MyBase.Enter

    Me.SelectAll()
End Sub
End Class
```

اختر الامر Build TextExProject من قائمة Build لتترجم المشروع إلى ملف من النوع DLL لتتمكن من استخدام هذه الأداة في مشاريع أخرى.

برمجة الأداة:

اختر الامر الفرعي New Project من الامر Add Project من القائمة File حتى تنشئ مشروع من النوع Windows Application في نفس الحل Solution، ولا تنسى جعله المشروع الابتدائي **Startup Project** (حتى يتم تنفيذه بعد الضغط على المفتاح [F5])، وذلك بالضغط على رمزه في نافذة مستكشف الحل Solution Explorer واختيار الامر Set as Startup Project من القائمة المنبثقة (شكل 2-16).



شكل 2-16: تحديد المشروع الابتدائي Startup Project.

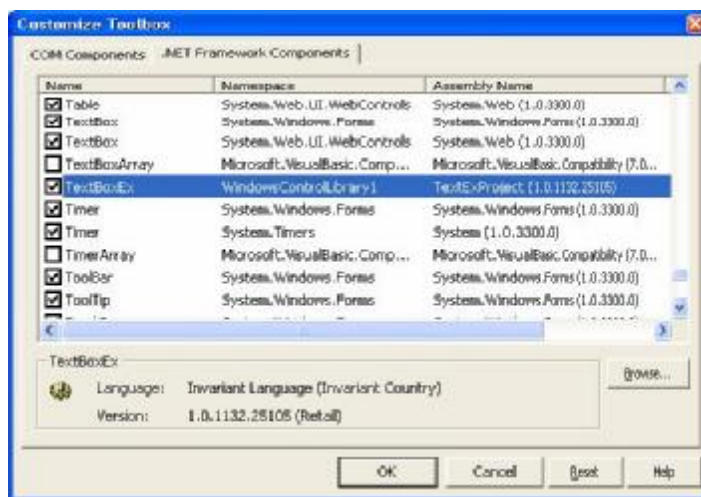
ملاحظة

بإمكانك إنشاء حل جديد New Solution إن اردت استخدام الأداة، ولكن يفضل إنشاء مشروع Windows Application في نفس الحل إن كنت لازلت تحت مرحلة التجربة واختبار الأداة، لتتمكن من الانتقال إلى شيفرة الأداة وتحريرها ومن ثم ترجمتها بسرعة.

ميزة أخرى في دمج المشاريع في نفس الحل يظهر جليا لحظة التنقيح Debugging، حيث ستتعامل مع أدوات التنقيح (والخاصة ببيئة Visual Studio .NET) وكأنك كلا المشروعين مشروع واحد، فقيم المتغيرات ومسارات التنفيذ وغيرها من الأمور، يمكنك تتبعها بين ملفات كلا المشروعين.

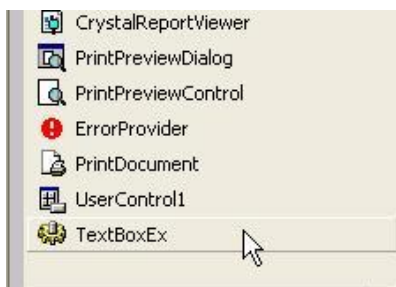
ميزة ثالثة، عندما تقوم بإعادة ترجمة الأداة، ستقوم بيئة التطوير بنسخ ملف مجمع الأداة المحدث إلى مجلد البرنامج الذي يستخدمها بشكل تلقائي.

بينما تعمل على نافذة مصمم النماذج، انقر بزر الفأرة الأيمن على صندوق الأدوات Toolbox واختر الأمر Customize Toolbox ليظهر لك صندوق حوار بعنوان Customize Toolbox، انتقل إلى خانة التثبيت NET Framework Component. واضغط على الزر Brows (في أسفل يمين صندوق الحوار)، ابحث عن ملف الأداة في نفس المسار الذي قممت بترجمة ملف الأداة فيه. وبعد إيجاده والضغط على الزر Open، ستلاحظ أنه تم تحديد الأداة في صندوق الحوار Customize Toolbox (شكل 16-3).



شكل 16-3: إضافة الأداة TextBoxExProject في صندوق الأدوات.

تأكد من تحديدك للأداة واضغط على الزر OK، ستلاحظ ان بيئة التطوير Visual Studio .NET قد اضافت رمز للأداة في صندوق الأدوات بالاسم TextBoxEx (شكل 16-4)، يمكنك الان استخدامها وإضافة نسخة منها في نافذة النموذج.



شكل 16-4: ظهور الأداة TextBoxEx في صندوق الأدوات.

تعامل مع الأداة TextBoxEx كما تتعامل مع الأداة TextBox التقليدية فهي مشتقة منها، الشيء الظريف الذي يثبت لنا تطوير الأداة هو ظهور الخاصية AutoSelect (والتي اضعناها في فئة الأداة الجديدة) بنافذة الخصائص (شكل 16-5).



شكل 16-5: ظهور الخاصية AutoSelect في نافذة الخصائص.

جرب اضافة ثلاث أو اربع أدوات TextBoxEx على نافذة النموذج، واسند القيمة True لكافة خصائصها، قم بنقل التركيز وقت التنفيذ بين الأدوات المختلفة، ستلاحظ انه سيتم تحديد كافة النص بمجرد حصول الأداة على تركيزها.

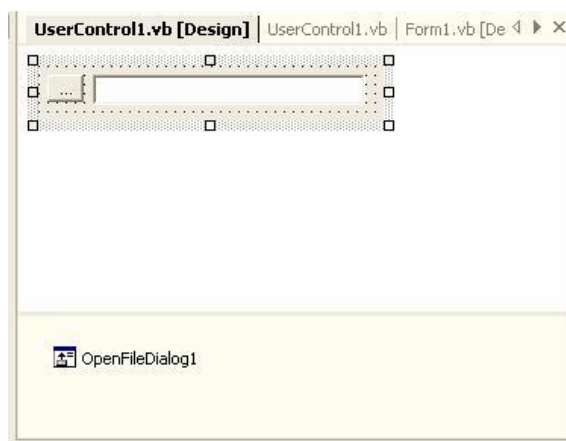
هذه ببساطة شديدة الفكرة من اتباع الاسلوب الاول لتصميم الأدوات الخاصة، وهو وراثة الأدوات لزيادة قوتها باضافة خصائص وطرق اضافية تناسب احتياجاتك. ولا تنسى انه يمكن للأداة المشتقة ان تصل إلى طرق وخصائص الأداة القاعدية Base Control، اصف إلى ذلك قدرتك على اعادة قيادة Overrides واعادة تعريف Overloads اعضاء الفئة القاعدية. الفقرة التالية تمكنك من تطوير أداة خاصة تحضن مجموعة من الأدوات الجاهزة.

حضن مجموعة من الأدوات

عليك معرفة، ان الأداة الخاصة التي تولفها ليست سوى نافذة نموذج Form تقليدية تمكنك من وضع الأدوات عليها وتعديل خصائصها وتصميمها بشكل مرئي Visual كما تفعل مع نوافذ النماذج، والاختلاف بينهما بسيط جدا لا يتعدى الخصائص والطرق، فالأداة الخاصة لا تحتوي على خصائص تتبع منطقيا إلى نوافذ النماذج (كالخصائص WindowState، ShowInTaskBar، MaximizeBox... الخ).

سأحاول في هذه الفقرة إعطائك مثلا يعرض لك كيفية الاستفادة من اسلوب حضن الأدوات لاختصار العمليات المتكررة، كعملية وضع زر Button مرافق لأداة TextBox يمكن المستخدم من الضغط عليها لكتابة اسم الملف.

أنشئ أداة خاصة Custom Control جديدة، وأضف على جبهتها ثلاث أدوات هي: Button، TextBox، و OpenFileDialog (شكل 6-16).



شكل 6-16: أداة خاصة حاضنة لمجموعة أدوات.

عدل الخاصية Anchor للأداة TextBox (بحيث تكون: فوق، يمين، ويسار)، وفي الحدث Click التابع للأداة Button، سطر الشيفرة التالية:



```
Private Sub Button1_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles Button1.Click

    If OpenFileDialog1.ShowDialog = DialogResult.OK Then
        TextBox1.Text = OpenFileDialog1.FileName
    End If
End Sub
```

والان يمكنك الانتقال من مرحلة التأليف إلى التصميم ووضع الأداة على نافذة النموذج، ودون الحاجة لكتابة شيفرة مصدريه واحدة، تستطيع ان تمكن المستخدم من كتابة اسم الملف في أداة النص بعد اختياره من صندوق حوار فتح.

نقاط وضعها في عين الاعتبار:

عند حضن مجموعة خارجية من الأدوات في أداتك الخاصة، عليك ان ترفق ملفات المجمعات Assemblies مع أداتك الخاصة إلى المبرمج الذي سيتعامل معها، ففي الأداة التي صممناها للتو، تستخدم مجموعة من الأدوات الجاهزة والتابعة لمجمعات إطار عمل .NET Framework، صحيح ان احتمالية وجود هذه ملفات هذه المجمعات كبير جدا في اجهزة المبرمجين، إلا انه يستحسن إبلاغهم بان أداتك الخاصة تحتاج إلى وجود هذه المجمعات، حتى يحاول المبرمجون توفير نسخ منها ليس في أجهزتهم وحسب بل حتى في اجهزة المستخدمين لبرامجهم.

بالنسبة لمحدد الوصول للأدوات المحضونة في مرحلة التأليف، فلا بد ان يكون Public حتى تمكن المبرمج من الوصول إلى أعضاء الأدوات المحضونة، وتذكر ان الأدوات المحضونة تكون محدثات وصلها Friend بشكل افتراضية، لذلك قد تحتاج إلى تغييرها من نافذة الشيفرة أو من الخاصية Modifiers في نافذة الخصائص.

صحيح ان اختيار محدد الوصول Public يريحك من عناء التصنيف الفرعي Subclassing لأعضاء الأدوات المحضونة، إلا انه سيتمكن المبرمج من الوصول إلى الأداة المحضونة وكأنه مؤلف الأداة الخاصة، وقد لا يقوم المبرمج بالتعامل مع الأدوات المحضونة بالشكل المناسب، الامر الذي قد يؤثر على كامل سلوك تنفيذ الأداة الخاصة وباقي الأدوات المحضونة بها. (تخيل ان قام المبرمج -مثلا- بتغيير موقع الأداة المحضونة TextBox في المثال السابق بطريقة الخطأ).

ملاحظة

اقصد بالتصنيف الفرعي لأعضاء الأدوات المحضونة في السياق السابق، عملية تعريف إجراءات عامة Public لأعضاء الأدوات التي لم تستخدم معها محدد الوصول Public، مثال في الأداة الخاصة السابقة:

```
Public Property FileName() As String
    Get
        Return Me.TextBox1.Text
    End Get
    Set(ByVal Value As String)
        Me.TextBox1.Text = Value
    End Set
End Property
```

إنشاء أداة مستقلة

في هذه الحالة، فانك لن تعتمد على أدوات أخرى لتصميم اداتك الخاصة، فلن تترك أداة أخرى ولن تحضن أداة أخرى، بل ستعتمد على كائنات GDI+ لترسم وتصمم واجهة الأداة.

عيوب هذا اسلوب تغطي على محاسنه، فعليك اعادة كتابة معظم الطرق والخصائص بنفسك، وستكون المسئول الاول والآخر عن كل صغيرة وكبيرة في الشيفرة المصدرية، كما ان الجهد والوقت الذي تبذله في انجاز الأداة يعادل تصميم برنامج يعتمد كامل يعتمد على أدوات جاهزة -أسأل مجرب ولا تسأل طبيب.

لا توجد ميزة تستطيع إعطائك إياها عن هذا الاسلوب إلا ميزة السيطرة والتحكم المطلق الذي تجنيه على كل جزء من أجزاء الأداة، فأنت ستكون مؤلف الشيفرة وهي ملكا لك.

مع ذلك، كلي ثقة وأمل بإخواني المبرمجين العرب في تطوير أدوات خاصة قوية وداعمة لبرامجنا العربية، خاصة بعد الدعم العربي القوي الذي وفرته لنا Microsoft في اطار عمل .NET Framework.

وكمثال على تطوير هذا النوع من الأدوات، سنقوم بإنشاء أداة بالاسم Rotator تمكن المبرمج من قلب النصوص بمقدار زاوية معينة (شكل 16-7)، تحتوي هذه الأداة على خاصيتين Text و Angle، الخاصية الثانية تمثل مقدار زاوية الدوران، وبالنسبة للنص الموجود في الخاصية Text فيمكن رسمه في الحدث Paint والخاص بالأداة:



```
Private Sub Rotator_Paint(ByVal sender As Object, _
    ByVal e As System.Windows.Forms.PaintEventArgs) Handles
    MyBase.Paint

    Dim gr As Graphics = e.Graphics
    Dim sf As New StringFormat()

    gr.TranslateTransform(Me.ClientRectangle.Width / 2, _
        Me.ClientRectangle.Height / 2)

    gr.RotateTransform(Me.Angle)

    gr.DrawString(Me.Text, Me.Font, Brushes.Black, New RectangleF(0, _
        0, Me.ClientRectangle.Width, Me.ClientRectangle.Height))

    gr.ResetTransform()
    gr.DrawRectangle(Pens.Black, Me.ClientRectangle)

End Sub
```



شكل 16-7: الأداة المستقلة Rotator.

ينقص الأداة Rotator شيء واحد تقريبا وهو إضافة شيفرات لتوسيط النص وسط الأداة بالضبط حتى يتم عرضها بالشكل الصحيح (ستحتاج إلى استخدام الدوال المثلثية $\text{Math.Sin}()$ و $\text{Math.Cos}()$ لعمل ذلك).

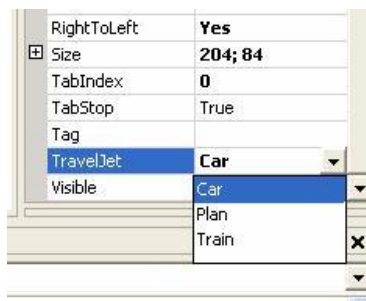
لمسات فنية إضافية

دعني أرشدك في هذه الفقرة إلى بعض اللمسات الفنية التي تضيفها على ادائك الخاصة لجعلها كباقي أدوات Windows Forms حقيقية وكأنها مصممة لأغراض تجارية. في البداية حاول استخدام التركيبات من النوع Enum دائما للخاصية ان كانت القيم التي تسندها اليها معينة:

```
Enum Flight
    Car
    Plan
    Train
End Enum

Property TravelJet() As Flight
    ...
End Property
```

الثمرة الابتدائية التي تجنبها من استخدام التركيبات من النوع Enum، هو التسهيل على المبرمج لاسناد قيمها سواء برمجيا من نافذة محرر الشيفرة، أو وقت التصميم عن طريق نافذة الخصائص (شكل 16-8).



شكل 16-8: ظهور قائمة تمثل القيم المختلفة للخاصية في نافذة الخصائص.

المزيد ايضا، حاول استخدام الأنواع المعرفة من البيانات ان كانت قريبة من الانواع البيانات الشهيرة والاكثر استخداما في مشاريع Windows Forms (كـ Size، Color، Font، Rectangle... الخ)، فذلك يسهل على المبرمج ايضا اسناد قيمها بطرق مختلفة (شكل 16-9 بالصفحة التالية):

```
Property TextColor() As Color
...
End Property
```



شكل 16-9: أضيف لوح ألوان للخاصية TextColor في نافذة الخصائص.

ملاحظة

إن كنت تنوي عرض حقول وخصائص فئاتك الخاصة في نافذة الخصائص بشكل شجري، عليك استخدام كائن من النوع TypeConverter -راجع مكتبة MSDN للحصول على مثال لاستخدامه.

بعض الأدوات تحتوي على مجموعة من الخصائص تسمى **خصائص وقت التصميم** **Design time properties**، وهي خصائص لا يمكن تعديل قيمها إلا وقت التصميم، تستطيع عمل ذلك باختبار قيمة الخاصية **DesignMode** والتي تعود بالقيمة **True** إن كانت الوقت هو وقت التصميم و **False** وقت التنفيذ:

```
Public Property ModeType() As Boolean
    Get
        If Not Me.DesignMode Then
            Throw New Exception("هذه الخاصية وقت التصميم فقط")
        End If
    End Get
    Set(ByVal Value As Boolean)
        If Not Me.DesignMode Then
            Throw New Exception("هذه الخاصية وقت التصميم فقط")
        End If
    End Set
End Property
```

ملاحظة

رغم ان اختبار قيمة الخاصية DesignMode تابعة لمرحلة التأليف Authoring time للأداة، إلا ان القيمة التي تعود بها تابعة لمرحلة التصميم Design time من قبل المبرمج وليس مؤلف الأداة.

مواصفات إضافية:

يوفر لك مجال الاسماء System.ComponentModel مجموعة كبيرة من المواصفات Attributes التي تعطيك تحكما اكثر من اعضاء أدواتك الخاصة، لديك مثلا المواصفة Description والتي تحدد فيها وصف نصي للخاصية يفيد المبرمج (شكل 10-16):

```
Imports System.ComponentModel
...
...
<Description("حدد لون النص الموجود في اعلى الأداة من هذه الخاصية")> _
Property TextColor() As Color
...
...
End Property
```



شكل 10-16: وصف الخاصية ظهر في اسفل نافذة الخصائص.

ان عرفت خصائص للقراءة فقط ReadOnly في اداتك الخاصة، فلا يوجد داعي من اظهارها على نافذة الخصائص، حيث ان المبرمج لن يتمكن من تعديل قيمها، اسند القيمة False إلى مشيد المواصفةBrowsable حتى يتم إخفاء الخاصية من نافذة الخصائص:

```

<Browsable(False)> _
ReadOnly Property IsSomething() As Boolean
    Get
        ...
        ...
        ...
    End Get
End Property

```

إذا كانت الخاصية حرفية أو أي نوع آخر تعتقد أنه يتأثر بالاعدادات الإقليمية، فمن المفضل إرسال القيمة True لمشيد الموصفة Localizable، حتى يتم حفظ نسخة من قيمة الخاصية لكل دولة النماذج المحلية Localized Forms في ملفات المصادر:

```

<Localizable(True)> _
Property TitleName() As String
    ...
    ...
End Property

```

الموصفتين DefaultProperty و DefaultEvent تحددان فيهما اسم الخاصية الافتراضية (التي يتم تحديدها في نافذة الخصائص بمجرد إنشاء نسخة من الأداة)، والحدث الافتراضي (الذي يتم اختياره في محرر الشيفرة لحظة النقر المزدوج وقت التصميم على الأداة:

```

<DefaultProperty("Text"), DefaultEvent("Click")> _
Public Class MyUserControl
    Inherits System.Windows.Forms.UserControl
    ...
    ...
End Class

```

أخيراً، استخدم الموصفة ToolboxBitmap لتحديد فيها الرمز أو الأيقونة التي تود أن تظهر بها الأداة في صندوق الأدوات Toolbox (اناسب حجم للصورة 16 x 16 بكسل):

```

<ToolboxBitmap ("C:\MyIcon.ico")> _
Public Class MyUserControl
    Inherits System.Windows.Forms.UserControl
    ...
    ...
End Class

```

تطوير خدمات Windows

في هذا القسم اعرض لك طريقة تطوير مشاريع تعرف بخدمات Windows Services.

ملاحظة

الفئات المستخدمة في هذا القسم مشمولة في مجال الأسماء التالي:

Imports System.ServiceProcess

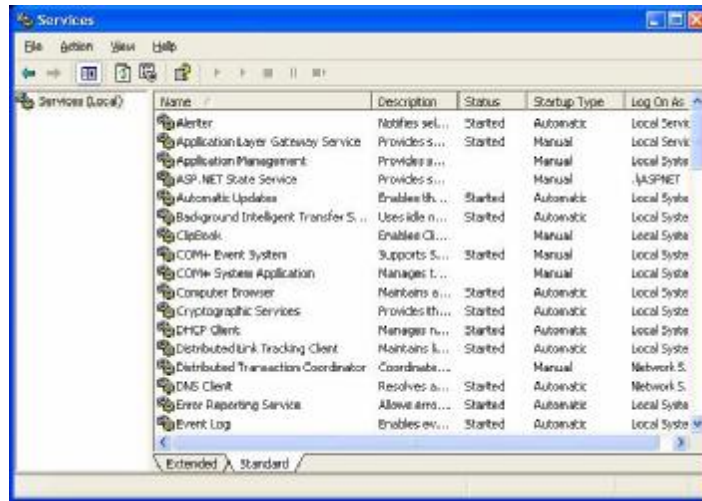
مع ذلك، معظم انواع المشاريع (كـ Windows Application) لا تدعم مجال الاسماء السابق، لذلك عليك استخدام صندوق حوار المراجع Add References وإضافة المجمع System.ServiceProcess.dll. اما المشاريع من النوع Windows Services فهي تشمل بشكل تلقائي.

مقدمة إلى خدمات Windows

خدمة Windows Service ما هي إلا برنامج تنفيذي ثنائي EXE يتم تشغيله بمجرد عملية اقلاع النظام System Booting - أي بمجرد بدء تشغيل نظام التشغيل. الفكرة من خدمات Windows موجه لتلك النوعية من البرامج التي تعمل فترة طويلة ولا تحتاج إلى ردة فعل من المستخدم Interaction، وفي الحقيقة معظم خدمات Windows لا تحتوي على واجهة استخدام User Interface، فالتوجه الذي تتبعه هو ان تعمل كخادم Server يقوم بمهام معينة.

توجد مجموعة كبيرة من خدمات Windows كخدمة Internet Information Server (المألوفة بالاختصار خادم IIS)، كخدمة Microsoft SQL Server، Microsoft Proxy Server وغيرها، الغرض منها تنفيذ مهام إدارية Administrating، توجيه Directing، مراقبة Watching،... الخ دون الحاجة لوجود مستخدم على الجهاز.

يمكنك معرفة جميع خدمات Windows الموجودة في جهازك الشخصي بالنقر المزدوج على الرمز Services من المجموعة Administrative Tools في لوحة التحكم Control Panel (شكل 11-16)

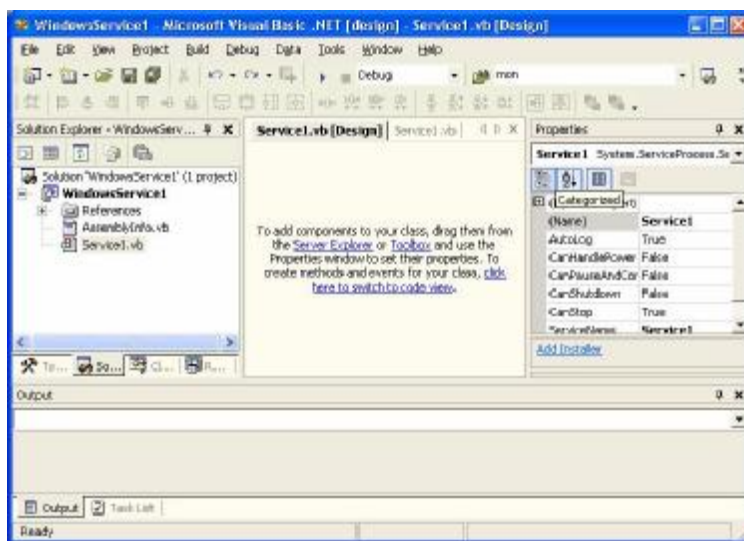


شكل 11-16: خدمات Windows الموجودة في الجهاز.

يمكنك الضغط بزر الفأرة الايمن على أي خدمة من الخدمات الموجودة في الجهاز واختيار امر من ثلاثة اوامر تميز خدمات Windows هي: **Start** لتنفيذ الخدمة، **Stop** لايقاف عمل الخدمة، و **Pause** للايقاف المؤقت للخدمة (ستحتاج إلى الضغط على **Resume** لتنفيذ الخدمة بعد الوقف المؤقت).

إنشاء مشاريع من النوع Windows Service

كل ما هو مطلوب منك اختيار الامر New->Project من قائمة File وتحديد الرمز Windows Service لإنشاء مشروع يعمل كخدمة Windows، ستتشى لك بيئة التطوير Visual Studio .NET مشروعا جديدا يحتوي على الملف Service1.vb (شكل 11-16).



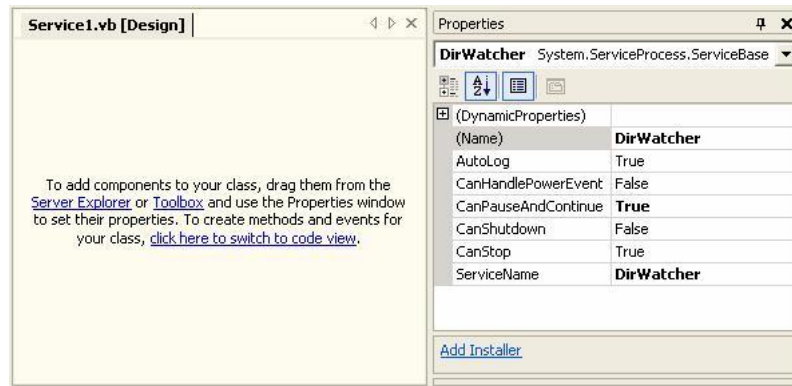
شكل 16-12: مشروع من النوع Windows Service.

خدمة Windows التي سنقوم ببنائها عبارة عن مراقبة مجلد النظام Windows ومعرفة الملفات التي تم حذفها منذ بداية تشغيل الخدمة (مع بداية تشغيل الجهاز)، حتى إنهائها، غير اسم المشروع من WindowsService1 إلى FolderWatcher.

في الفقرة السابقة عرفنا على معنى خدمة Windows بمنظور المستخدم، لذلك دعني أعيد صياغة التعريف بمنظور مبرمج .NET. والذي يعرف خدمة Windows على أنها فئة Class مشتقة من الفئة القاعدية System.ServiceProcess.ServiceBase.

من التعريف السابق نستنتج ان المشروع الذي أنجزناه للتو يحتوي على خدمة Windows باسم Service1، انتقل إلى نافذة الخصائص وعدل الخاصيتين (Name) و ServiceName من DirWatcher إلى Service1.

اسنادك للقيمة True للخاصية CanStop تحدد فيها ما اذا كانت الخدمة قابلة للتوقف Stop (من لوحة التحكم Control Panel أو أي برنامج اخر)، اما الخاصية CanPauseAndContinue فتحدد فيها قابلية الايقاف المؤقت للخدمة (شكل 16-13).



شكل 13-16: تعديل خصائص الخدمة.

تصحيح الشيفرة

ان قمت بفتح نافذة محرر الشيفرة للخدمة DirWatcher السابقة، سترى هذه الشيفرة المولدة:

```
Imports System.ServiceProcess

Public Class DirWatcher
    Inherits System.ServiceProcess.ServiceBase

    #Region " Component Designer generated code "

    Public Sub New()
        MyBase.New()

        ' This call is required by the Component Designer.
        InitializeComponent()

        ' Add any initialization after the InitializeComponent() call

    End Sub

    'UserService overrides dispose to clean up the component list.
    Protected Overrides Sub Dispose(ByVal disposing As Boolean)
        If disposing Then
            If Not (components Is Nothing) Then
                components.Dispose()
            End If
        End If
        MyBase.Dispose(disposing)
    End Sub

    ' The main entry point for the process
```

```

<MTAThread()> _
Shared Sub Main()
    Dim ServicesToRun() As System.ServiceProcess.ServiceBase

    ' More than one NT Service may run within the same process. To
add    ' another service to this process, change the following line
to    ' create a second service object. For example,
    '
    '     ServicesToRun = New System.ServiceProcess.ServiceBase (
{New Service1, New MySecondUserService}
    '

    ServicesToRun = New System.ServiceProcess.ServiceBase() {New
Service1() }

    System.ServiceProcess.ServiceBase.Run(ServicesToRun)
End Sub

' Required by the Component Designer
Private components As System.ComponentModel.IContainer

' NOTE: The following procedure is required by the Component
Designer
' It can be modified using the Component Designer.
' Do not modify it using the code editor.
<System.Diagnostics.DebuggerStepThrough()> Private Sub
InitializeComponent()
    '
    'DirWatcher
    '
    Me.CanPauseAndContinue = True
    Me.ServiceName = "DirWatcher"

End Sub

#End Region

Protected Overrides Sub OnStart(ByVal args() As String)
    ' Add code here to start your service. This method should set
things    ' in motion so your service can do its work.
End Sub

Protected Overrides Sub OnStop()
    ' Add code here to perform any tear-down necessary to stop
your service.
End Sub

End Class

```

الجزء المكتوب بالخط السميك Bold في الشيفرة السابقة، يتطلب منك تتقيحه يدويا (حيث لم يظهر فيه تأثير تغيير الخاصية ((Name)، عدل الكلمة من Service1 إلى DirWathcer:



```
ServicesToRun = New System.ServiceProcess.ServiceBase() {New
DirWathcer() }
```

بهذا نكون قد صححنا الخطأ وأصبحنا جاهزين لكتابة شيفرات الخدمة، ولكن قبل ذلك دعني أعرفك على الفئة FileSystemWatcher.

الفئة System.IO. FileSystemWatcher

لست هنا بصدد شرح كافة تفاصيل الفئة FileSystemWatcher فهي لا تتعلق بموضوع خدمات Windows بشكل مباشر، ولكن بودي عرض طريقة استخدامها بشكل سريع حيث انني سأستخدمها في مشروع الخدمة FolderWatcher الذي صممناه. الغرض الأساسي من الفئة FileSystemWatcher يتضح من اسمها في مراقبة الملفات، حيث تحدد في خاصيتها Path مسار المجلد الذي تود مراقبة ملفاته:

```
Dim FW As New System.IO.FileSystemWatcher()
FW.Path = "C:\Windows"
```

أي تعديل في ملفات هذا المجلد (حذف، اضافة، أو تعديل خصائص ملفاته) سيتم إبلاغك به فوراً عن طريق مجموعة من الاحداث توفرها لك الفئة كـ Created، Deleted، Changed، و Renamed (عليك قنص الاحداث اما بـ WithEvents أو AddHandler بنفسك). ترسل هذه الاحداث مع وسيطتها كائن من نوع FileSystemEventArgs يحتوي على مجموعة من الخصائص منها FullPath التي تعود بالمسار الكامل للملف الذي طرأ عليه التعديل:

```
Dim WithEvents FW As New System.IO.FileSystemWatcher()
Public Sub FW_Deleted(ByVal sender As Object, _
    ByVal e As System.IO.FileSystemEventArgs) Handles FW.Deleted
    MsgBox (e.FullPath)
End Sub
```

حتى يتم تنفيذ الاحداث السابقة في الوقت المناسب، عليك اسناد القيمة True لخاصيته EnableRaisingEvents لبدء عملية مراقبة المجلد، وقد تسند القيمة True ايضا للخاصية IncludeSubdirectories لتشمل المراقبة المجلدات الفرعية:

```
FW.IncludeSubdirectories = True
FW.EnableRaisingEvents = True
```

هذا كل ما تحتاج معرفته حول الفئة System.IO.FileSystemWatcher لتطبيقه في خدمة Windows التي نصممها، ان اردت مزيد من التفاصيل حول هذه الفئة يمكنك مراجعة مكتبة MSDN.

كتابة الشيفرات

تحتوي الفئة القاعدية System.ServiceProcess.ServiceBase على طريقتين هما OnStart() و OnStop()، يتم تنفيذ الاولى لحظة تشغيل الخدمة والثانية لحظة إيقافها، عليك اعادة قيادة Overrides هاتين الطريقتين حتى تضع الشيفرات المتطلب تنفيذها لحظة بداية تنفيذ وإيقاف الخدمة:



```
Public Class DirWatcher
    Inherits System.ServiceProcess.ServiceBase
    ...
    ...
    Dim WithEvents FW As New System.IO.FileSystemWatcher()

    Protected Overrides Sub OnStart(ByVal args() As String)
        FW.Path = "C:\Windows"
        FW.IncludeSubdirectories = True
        FW.EnableRaisingEvents = True
    End Sub

    Protected Overrides Sub OnStop()
        FW.EnableRaisingEvents = False
    End Sub
End Class
```

بما اننا أسندنا القيمة True للخاصية CanPauseAndContinue (شكل 16-13)، فعلينا اعادة قيادة الطريقتين OnPause() و OnContinue() حتى نصف الشيفرات المطلوب تنفيذه لحظة الإيقاف المؤقت Pause وإكمال التنفيذ بعد الإيقاف المؤقت Resume:



```
Public Class DirWatcher
    Inherits System.ServiceProcess.ServiceBase
    ...
    ...
    Protected Overrides Sub OnPause()
        FW.EnableRaisingEvents = False
    End Sub

    Protected Overrides Sub OnContinue()
        FW.EnableRaisingEvents = True
    End Sub
End Class
```

الخطوة الأخيرة هي قنص احدث الكائن FileSystemWatcher لتحديد ماذا نريده من الخدمة ان تفعل عندما يتم حذف ملف من ملفات المجلد Windows. الشيفرة التالية ستقوم بكتابة اسم الملف الذي تم حذفه في ملف نصي Test.TXT (لا تتسى ان خدمات Windows لا تظهر أي واجهة استخدام، حيث لا يفترض وجود شخص على جهاز الخادم Server):



```
Public Class DirWatcher
    Inherits System.ServiceProcess.ServiceBase
    ...
    ...
    Dim WithEvents FW As New System.IO.FileSystemWatcher()

    Private Sub FW_Deleted(ByVal sender As Object, _
        ByVal e As System.IO.FileSystemEventArgs) Handles FW.Deleted

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

        textFile.WriteLine("تم حذف الملف" & e.FullPath)

        textFile.Close()
    End Sub
End Class
```

انظر ايضا

لمزيد من التفاصيل حول التعامل مع وحدات التخزين Streams والملفات النصية، قد تحتاج إلى قراءة الفصل الثامن **الملفات والمجلدات**.

تسجيل الخدمة

ان استعجلت وحاولت تنفيذ الخدمة (بالضغط على المفتاح [F5])، ستظهر لك بيئة التطوير .NET Visual Studio رسالة خطأ (شكل 14-16) مفادها ان الخدمة لا يمكن ان يتم تنفيذها كما تفعل مع تطبيقات EXE الاخرى، اذ عليك تركيبها وتثبيتها في الجهاز ليتم تسجيلها في النظام ومن ثم يمكنك تنفيذها.



شكل 14-16: رسالة خطأ مفادها ان الخدمة لا يمكن تنفيذها دون تسجيلها.

حتى تمكن الخدمة من التركيب، عليك كتابة مجموعة من الشيفرات الإضافية لتعريف كائنات من فئات الخاصة بتثبيت وتسجيل الخدمة، مع ذلك لست بحاجة لفعل ذلك يدوياً، حيث يمكن لبيئة التطوير .NET Visual Studio من توليد الشيفرة تلقائياً نيابة عنك. اضغط على الرابط المعنون **Add Installer** في اسفل نافذة الخصائص والخاصة بفئة الخدمة (شكل 13-16 صفحة 590)، ستلاحظ أن بيئة التطوير أنشأت لك ملف `ProjectInstaller.vb` يحتوي على أداتين هما `ServiceInstaller1` و `ServiceProcessInstaller1` (شكل 15-16).



شكل 15-16: ملف تثبيت الخدمة.

حدد الأداة الاولى `ServiceInstaller` وانتقل إلى نافذة الخصائص، ستلاحظ وجود مجموعة من الخصائص والخاصة بها، يمكنك استكشاف معانيها من مستندات .NET Documentation.

حيث لا يهمني في الوقت الحالي إلا الخصائص ServiceName, DisplayName, و StartType، الأولى لابد ان يكون اسم الخدمة مطابق للاسم الخدمة المراد تثبيتها (في حالتنا ستكون DirWatcher)، الثانية هو الاسم الظاهري للخدمة (اكتب مثلا- "خدمة DirWatcher")، وبالنسبة للخاصية الثالثة فيمكنك اسناد القيمة Automatic ان اردت تنفيذ الخدمة بمجرد اقلاع النظام System Booting (يعني بمجرد تشغيل نظام التشغيل).

بالنسبة للأداة الأخرى ServiceProcessInstaller1 فعليك تحديد خاصيتها Account لتحديد نوع حساب المستخدم User Account الذي تعمل الخدمة فيه، اسند القيمة LocalSystem لها والتي تتجاهل حساب المستخدم وتعمل لاي مستخدم سواء في النظام الحالي أو على مستوى الشبكة المحلية. (راجع مكتبة MSDN لمزيد من التفاصيل حول القيم الأخرى).

قم الان بعملية ترجمة ملفات مشروع الخدمة ليتم توليد ملف ثنائي بالامتداد EXE، ستحتاج اليه عند استخدام الأداة InstallUtil.EXE والتي تقوم بالتثبيت الفعلي للخدمة بمسجل نظام التشغيل System Registry.

الأداة InstallUtil.EXE

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\InstallUtil.EXE) حيث يقوم بتحميل مسارات Paths الأداة InstallUtil.EXE ويسهل عليك الوصول لها، مع ذلك لست بحاجة إلى تنفيذ هذا الملف ان كنت قد شغلت نافذة موجه الاوامر من خلال الرمز Visual Studio .NET Command Prompt الموجود في المجموعة البرمجية Microsoft Visual Studio .NET بقائمة Start (تماما مثل ما فعلنا في الفصل الحادي عشر **المجموعات Assemblies** عندما استخدمنا أدوات الترجمة، الربط، والتسجيل) (شكل 11 - 6).

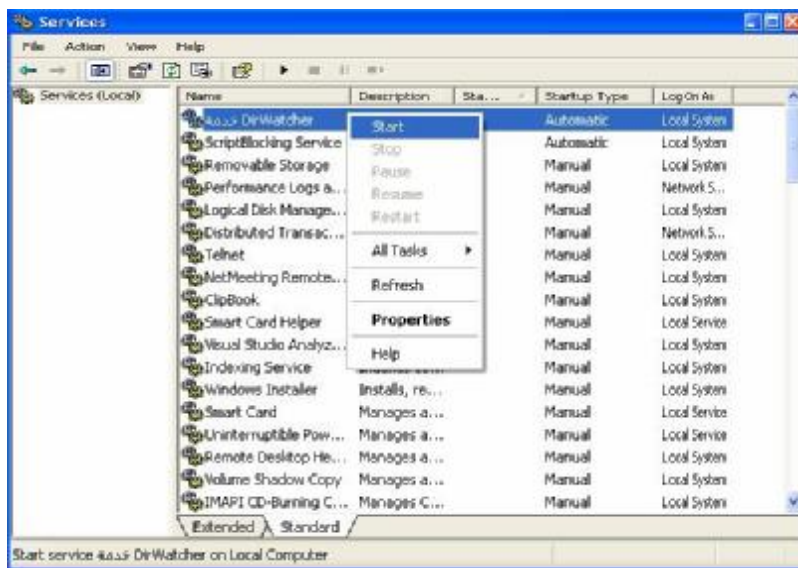
لتسجيل خدمتنا DirWatcher، اكتب اسم ملف الخدمة التنفيذي EXE مع الأداة

:InstallUtil

```
C:\>InstallUtil FolderWatcher.EXE
```

ان تم كل شيء على ما يرام، سيتم تشغيل الخدمة بمجرد بدء اقلاع النظام، مع ذلك لست بحاجة إلى اعادة تشغيل الجهاز Restart لعمل ذلك، فتستطيع الانتقال إلى قائمة الخدمات Service في

المجموعة Administrative Tools في لوحة التحكم Control Panel، والضغط بزر الفأرة الايمن على خدمتنا ثم اختيار الامر Start من القائمة المنبثقة (شكل 16-16).



شكل 16-16: تشغيل خدمة DirWatcher من قائمة الخدمات في لوحة التحكم.

اخيرا، ان اردت الغاء تنبيات الخدمة وتسجيلها من النظام، استخدم نفس الأداة InstallUtil.EXE بنفس الصيغة السابقة، ولكن مع ارسال المدخل /u:

```
C:\>InstallUtil FolderWatcher.EXE /u
```

ملاحظة

عليك ايقاف عمل الخدمة Stop قبل الغاء تثبيتها. بالنسبة لخدمتنا DirWatcher فيمكنك ايقافها من قائمة الخدمات (شكل 16-16) وذلك باختيار الامر Stop من القائمة المنبثقة التي ستظهر بنقر زر الفأرة الايمن عليها.

فئات أخرى

ونحن على مشارف الانتهاء من عالم تطوير تطبيقات Windows، بودي عرض مجموعة من الفئات التي قد تحتاجها عند تطوير برامجك العاملة تحت نظم Windows، سواء كانت تطبيقات قياسية Windows Application، خدمات Windows Services، أو حتى أدوات خاصة Custom Controls.

الفئة Application

تمثل الفئة Application المجمع الحالي والذي يتم تنفيذه الآن، ولا يمكنك إنشاء نسخة كائن منها باستخدام New، فكل مجمع يحتوي على كائن Application واحد. معظم خصائص وطرق الفئة Application مشتركة Shared Members، كما انها للقراءة فقط ReadOnly، كالخاصية ExecutablePath التي تعود بالمسار الكامل لملف المجمع الرئيسي، والخاصية StartupPath التي تعود بالمسار دون الملف:

```
MsgBox (Application.ExecutablePath) ' C:\Folder\MyAss.EXE
MsgBox (Application.StartupPath) ' C:\Folder
```

من الخصائص التي تعود بمعلومات حول المجمع: الخاصية CompanyName التي تعود باسم الشركة المضمونة في المجمع، الخاصية CurrentCulture التي تعود بالاعدادات الاقليمية للمجمع، الخاصية ProductVersion التي تعود بالاصدار، والخاصية ProductName التي تعود باسم المجمع.

اما الطرق، فتوجد الطريقة DoEvents التي توزع وقت المعالجة في نفس مسار التنفيذ احداث الأدوات المختلفة، الطريقة Exit() التي تنهي عمل المجمع بينما الطريقة ExitThread() تغلق جميع النوافذ التي تعمل في مسار التنفيذ الحالي.

راجع مكتبة MSDN لمزيد من التفاصيل حول اعضاء الفئة Application، وذلك لانني سأختم هذه الفقرة بذكر ثلاث احداث منها هي: ApplicationExit، ThreadExit، و Idle يتم تنفيذها بمجرد انتهاء البرنامج، انتهاء مسار تنفيذ، أو ان البرنامج في حالة الاستقرار Idle (أي لا توجد رسائل نظام أو أي مهام في طابور الرسائل يتطلب تنفيذها).

الفئة Cursor

النماذج والأدوات تحتوي على الخاصية Cursor والتي لم اتطرق لها في الفصول السابقة لحاجة ما في نفس يعقوب وربطها بالفئة Cursor في هذه الفقرة.

قبل ان ابدأ بالتحدث عن الفئة Cursor دعني اتحدث عن الخاصية Cursor والتابعة للأدوات الفئات، يمكنك اسناد قيمة إلى هذه الخاصية تمثل شكل مؤشر الفأرة من 28 شكل توفره لك الخاصية. يمكنك تخصيص شكل المؤشر عند مروره فوق كل أداة من الأدوات عن طريق خاصية الأداة Cursor:

```
Button1.Cursor = Cursors.No
```

اما ان اردت تخصيص أشكال مشيرة من عندك، فلن تجد اسهل من استخدام الفئة Cursor والتي يمكنك تحميل ملف المؤشر بإرساله إلى مشيدها:

```
Dim myCur As New Cursor ("C:\myCur.cur")
Button1.Cursor = myCur
```

ولا تنسى قتل كائن المؤشر عند عدم الحاجة اليه باستدعاء الطريقة Dispose():

```
myCur.Dispose()
```

المزيد ايضا، بدلا من تعيين شكل المؤشر لكل أداة على حده، يمكن اسناد قيمة إلى الخاصية المشتركة Current والتابعة للفئة Cursor حتى يتغير شكل المؤشر في كافة أدوات ونوافذ البرنامج:

```
Cursor.Current = Cursors.Hand
```

مع ذلك، ان كانت قيمة الخاصية Cursor والتابعة للأداة أو نافذة النموذج لا تساوي Default، فسيتم تغيير المؤشر عن المرور على الأداة أو النافذة بنفس القيمة الموجودة في خاصية الأداة أو النافذة Cursor.

لا ينحصر استخدام الفئة Cursor لتغيير شكل المؤشر، بل يشمل ايضا حكر المؤشر على نافذة أو أداة معينة، يمكنك عمل ذلك باسناد المنطقة إلى الخاصية المشتركة Clip (تتطلب كائن من النوع Rectangle)، الشيفرة التالية تحكر الفأرة في المنطقة الداخلية لنافذة النموذج:

```
Cursor.Clip = Me.RectangleToScreen(Me.ClientRectangle)
```

ولا تنسى تحرير الفأرة بإسناد القيمة Nothing إلى الخاصية Clip السابقة.

ملاحظة

الغرض من الطريقة RectangleToScreen() والتي استخدمتها في الشيفرة الأخيرة- هو الحصول على المنطقة بالنسبة للشاشة وليس النموذج.

أخيراً، لديك الخاصية المشتركة Position والتي يمكنك من تحريك مؤشر الفأرة إلى نقطة من الشاشة:

```
Cursor.Position = New Point(0, 0)
```

الفئة SendKeys

تتمكن الفئة SendKeys من محاكاة لوحة المفاتيح Keyboard والضغط على المفاتيح، تستطيع تحديد المفاتيح التي ترغب بتفعيلها باستخدام الطريقة المشتركة Send(). الشيفرة التالية تقوم بإرسال حروف اسمي ومن ثم الضغط على المفتاح [Enter]:

```
SendKeys.Send("~تركي")
```

الرمز السابق ~ يمثل مفتاح [Enter]، الرمز + مفتاح [Shift]، الرمز ^ مفتاح [Ctrl]، والرمز % مفتاح [Alt]، وإن اردت استمرار الضغط على المفتاح [Shift] -مثلاً- وإرفاقه بمفتاح آخر، استخدم الاقواس، فالشيفرة التالية ستكتب الحروف كبيرة Capital:

```
SendKeys.Send("+(turki)")
```

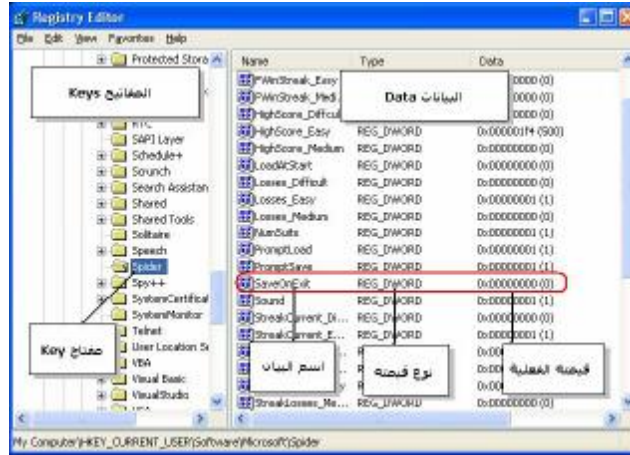
معظم المفاتيح الغير مطبوعة الأخرى تستخدم الاقواس المعكوفة { } مثل {TAB}، {ESC}، {LEFT}، {RIGHT}، {F1}، {F2}، {F3}،.... الخ، راجع مستندات .NET Documentation لمزيد من التفاصيل.

ملاحظة

لا يمكن استخدام الطريقة Send() لتنشيط برنامج آخر، والحل يتم إما يدويا بتنشيط ذلك التطبيق بالفأرة أو استخدامات إجراءات API كـ FindWindow و SetForegroundWindows.

الفتتان Registry و RegistryKey

تمتلك الفتتان Registry و RegistryKey من الوصول إلى مسجل النظام Windows Registry وقراءة محتوياته. وحتى يسهل عليك التعامل مع هاتين الفتتين، يفضل فهم واستيعاب تركيب مسجل النظام (شكل 16-17).



شكل 16-17: تركيب مسجل النظام Windows Registry.

يحتوي مسجل النظام على مجموعة كبير جدا من المفاتيح Keys أشبه ما توصف به بمجلدات، كل مفتاح من هذه المفاتيح يحتوي على مجموعة من القيم -أشبه بالخصائص، كل قيمة من هذه القيم تسمى بيان Data، وكل بيان يحتوي على قيمة تسمى Value. ان اردت التعامل مع الفتتين Registry و RegistryKey فالخطوة الاولى هي بالحصول على مرجع للمفتاح الجذري للمسجل، يمكنك الحصول على سبع مفاتيح جذرية عن طريق سبع خصائص مشتركة للفتة Registry والتي تعود بكائن من النوع RegistryKey:

```
Dim root1 As RegistryKey = Registry.ClassesRoot
Dim root2 As RegistryKey = Registry.CurrentConfig
Dim root3 As RegistryKey = Registry.CurrentUser
Dim root4 As RegistryKey = Registry.DynData
Dim root5 As RegistryKey = Registry.LocalMachine
Dim root6 As RegistryKey = Registry.PerformanceData
Dim root7 As RegistryKey = Registry.Users
```

ملاحظة

كلا الفئتين Registry و RegistryKey مشمولتان في مجال الاسماء Microsoft.Win32، لذلك قد تقوم باستيراده لاختصار كتابة اسماء انواع البيانات في شيفراتك المصدرية:

```
Imports Microsoft.Win32
```

الكائنات المنشأة من الفئة RegistryKey ستحتوي على ثلاث خصائص هي: Name اسم المفتاح، SubKeyCount عدد المفاتيح الفرعية للمفتاح الحالي، عدد البيانات التي يحتويها المفتاح الحالي.

يمكنك استكشاف المفاتيح الفرعية باستدعاء الطريقة GetSubKeyNames() والتي تعود بمصفوفة حرفية تشمل جميع المفاتيح الفرعية:

```
Dim name As String
For Each Name In root1.GetSubKeyNames
    ...
Next
```

ان كنت على دراية باسم المفتاح الفرعي، فيمكنك الوصول اليه مباشرة باستدعاء الطريقة OpenSubKey() لفتح ذلك المفتاح والعودة بكائن من النوع RegistryKey ايضا:

```
Dim IE As RegistryKey
IE = root1.OpenSubKey("Internet Explorer.Main")
```

ان فتحت كائن بالطريقة السابقة، عليك اغلاقه دائما:

```
IE.Close()
```

اما ان اردت تعديل قيمة بيانات المفتاح، فاستخدم الطريقة السابقة (`OpenSubKey()` مع ارسال القيمة `True` لفتح المفتاح للكتابة التي تتم باستدعاء الطريقة (`SetValue()` والتي تحدد فيها اسم البيان المراد تعديل قيمته، والقيمة المراد إسنادها له:

```
Dim IE As RegistryKey

IE = root1.OpenSubKey("Internet Explorer.Main")
Dim IE As RegistryKey

IE = root2.OpenSubKey("Internet Explorer.Main", True)
IE.SetValue("Search Page", "http://www.dev4arabs.com")
```

كما يمكنك استدعاء الطريقة (`GetValue()` لقراءة قيمة البيان.

الفئة Help

يمكنك إنشاء ملفات التعليمات `Help Files` باستخدام مجموعة من البرامج الجاهزة -لعل أبرزها `Microsoft HTML Help Compiler` - والتي تنتج ملفات من النوع `CHM`، تستطيع الوصول إلى احد صفحات ملف التعليمات عن طريق الفئة `Help` (لم اتحدث عن بناء ملفات التعليمات في هذا الكتاب).

تحتوي هذه الفئة على طريقتين مشتركة `Shared Methods`، الطريقة الاولى `ShowHelpIndex()` تعرض صفحة الفهرس `Index` (تتطلب وسيطة تمثل مرجع النافذة التي تتبع لها):

```
Help.ShowHelpIndex(Me, "C:\helpfile.chm")
```

اما الطريقة الاخرى `ShowHelp()` فتم اعادة تعريفها `Overloads` باربع صيغة، الصيغة الاولى تفتح لك القسم الخاص بعرض المحتويات `Contents`:

```
Help.ShowHelp(Me, "C:\helpfile.chm")
```

والصيغة الثانية ترسل وسيطة اضافية تحدد القسم الذي توده، كخانة تبويب البحث `Search`:

```
Help.ShowHelp(Me, "C:\helpfile.chm", HelpNavigator.Find)
```

ان ارسلت القيمة HelpNavigator.Topic إلى الوسيطة الثالثة في الصيغة السابقة، عليك ارفاقها برقم الموضوع في ملف التعليمات لتستخدم الصيغة الثالثة للطريقة ShowHelp():

```
Help.ShowHelp(Me, "C:\helpfile.chm", HelpNavigator.Topic , 3)
```

اما الصيغة الاخيرة فتمكنك من ارسال كلمة محجوزة keyword عرفت في ملف التعليمات لحظة إنشائه:

```
Help.ShowHelp(Me, "C:\helpfile.chm", "برمجة ")
```

يدعمك إطار عمل NET Framework. بمئات الفئات التي تسهل حياتك البرمجية لتطوير تطبيقات Windows، في أربعة فصول حاولت تلخيص جولتي حول ما قرأته من مستندات NET Documentation.. مع ذلك، ينقصك جزء هام وكبير في حياتك البرمجية مع إطار عمل NET Framework. وهو موضوع الجزء الرابع برمجة قواعد البيانات.

الجزء الرابع

برمجة قواعد البيانات

استخدام ADO.NET

عند الحديث عن قواعد البيانات، يتعالى صدى الحروف ADO.NET بين أحاديث المبرمجين، فهي الوسيلة المثلى لربط شيفراتهم البرمجية بملفات قواعد البيانات سواء كانت محلية أو على بعد آلاف الكيلومترات.

هذا الفصل هو مدخلك المبدئي إلى ADO.NET واستخدام فئاتها، ودعني انوه هنا بان هذا الكتاب مختص ببرمجة قواعد البيانات وليس التعامل مع قواعد البيانات بشكل عام، فلن اتحدث عن طريقة بناء الجداول، اعداد مخططات Entity Relation Diagrams (ERD)، أو ادارة نظم قواعد البيانات، بل سيكون حديثي محصور حول طرق الوصول إلى قواعد البيانات من شيفراتك المصدرية والمكتوبة بـ Visual Basic .NET.

ملاحظة

الغالبية الساحقة من فئات هذا الفصل والفصول التي تليه مشمولة في مجالات الاسماء التالية:

Imports System.Data
Imports System.Data.OleDb
Imports System.Data.SqlClient

فلا تنسى استيرادها في اعلى ملفات شيفراتك البرمجية، أو من صندوق حوار خصائص المشروع Project Property Pages لاختصار كتابة الشيفرات البرمجية.

مدخلك إلى ADO.NET

ADO.NET هي مجموعة من الفئات مشمولة في مجال الاسماء System.Data غرضها الوصول إلى مصادر البيانات **Data Sources**، والتي تمثل بيانات محفوظة تحت أنظمة قواعد بيانات متعددة الأنواع، الأجناس، والأعراق مما يعني قدرتك على الوصول إلى أي قاعدة بيانات مهما كانت الشركة المنتجة لها.

الوضع المتصل والوضع المنفصل

مهما كان نوع مصدر البيانات الذي تتعامل معه، عند استخدام ADO.NET عليك اختيار وضع من وضعين هما: **الوضع المتصل Connected Mode**، **والوضع المنفصل Disconnected Mode**.

في الوضع المتصل يتم الاتصال مع مصدر البيانات وتجري كافة العمليات -افضل ما أوصفها- على الهواء المباشرة، أو كما انك تجري محادثة هاتفية مع احد الأشخاص، فالإتصال مستمر وحي يبرز بين شيفراتك المصدرية وبين مصدر البيانات الأساسي، وأي خلل في مصدر البيانات أو إيقاف مؤقت، سيؤدي إلى خلل في شيفراتك المصدرية.

اما في الوضع المنفصل فهو قريب من فكرة صندوق الوارد **Inbox**، حيث تأتيك البيانات وتعدل فيها كما تشاء ومن ثم ترسل كافة التعديلات. وهذا ما يحدث مع ADO.NET بالضبط، فيعد قيامك بإجراء اتصال مع مصدر بيانات، ستحصل على البيانات المطلوبة وتنتهي علاقتك بمصدر البيانات الأساسي، وتتعامل مع البيانات -التي عندك- كما تتعامل معه الوضع المتصل. ليس هذا فقط، بل يمكنك تعديلها وتحريرها وإجراء كافة جمل الاستعلام **SQL** عليها، أو -الأكثر من ذلك- تعيد هيكلة البيانات **Data Structure** كتغيير في علاقات جداولها **Table Relationships** أو حقولها.

ميزة عظيمة في الوضع المنفصل يظهر من خلال استقلالية البيانات عن هيئة مصدر البيانات، ماذا يعني هذا؟ يعني انك تستطيع الحصول على بيانات من مصدر بيانات من النوع **Microsoft SQL Server®**، وتتعامل مع البيانات وتحريرها، ومن ثم ارسال البيانات إلى مصدر بيانات من النوع **Microsoft Access®**.

كان الغرض الأساسي من تطوير فكرة الوضع المنفصل هو تخفيف الضغط على مصادر البيانات في أجهزة الخادم **Servers** والتي تستخدم من قبل عدد كبير من العملاء **Clients**،

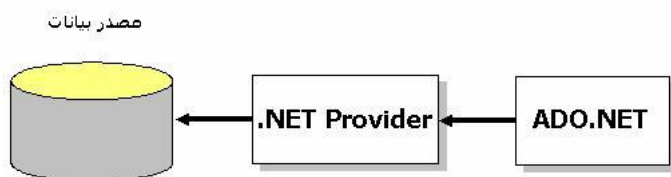
فيمكنك الحصول على البيانات قبيل ساعات الدوام، ومن ثم إرسالها وقت الغداء وهو انسب وقت يخف الضغط على الخادم فيه.

في هذا الفصل سيتمحور حديثي عن الوضع المتصل، حيث يعتبر مفتاحك لاستخدام والتعامل مع مصادر البيانات في الوضع المنفصل والذي سأنتطرق اليه في الفصل القادم ADO.NET للاتصال المنفصل.

مزودات .NET Data Providers

الميزة العظيمة التي تجنيها من استخدام ADO.NET هو استقلالية شيفراتك البرمجية عن الهياكل المختلفة لمصادر البيانات، أي ان نفس الشيفرات المصدرية التي استخدمتها لتطوير تطبيقات معتمدة على مصادر بيانات من نوع Microsoft Access®، ستستخدمها ايضا مع انواع مصادر بيانات اخرى كـ Oracle®، Microsoft SQL Server®، Microsoft FoxPro®... الخ، دون الحاجة لتغيير شيفراتك المصدرية.

السؤال الذي يطرح نفسه، كيف يمكن لـ ADO.NET من فعل ذلك رغم انها كتبت مرة واحدة فقط، وبالتالي ستكون موجه إلى نوع معين من مصادر البيانات؟ والجواب هو ان ADO.NET في الحقيقة لا تصل مباشرة إلى مصادر البيانات وانما تعتبر واجهة للمبرمج فقط، حيث انها تستخدم مزودات .NET Data Providers. (شكل 17-1).



شكل 17-1: مزودات .NET Providers تعمل كحلقة وصل بين ADO.NET ومصادر البيانات

مزودات .NET Data Providers ما هي الا حلقة وصل بين ADO.NET ومصادر البيانات، فعندما تريد من ADO.NET استخدام مصدر بيانات من النوع Microsoft

Access®، عليك توفير مزود NET Data Provider. والخاص بـ Microsoft Access®، اما ان كبر حجم قاعدة البيانات واردت الاعتماد على خادم كـ Microsoft SQL Server®، فكل ما هو مطلوب منك استخدام المزود الخاص بـ Microsoft SQL Server®. من منظور كمبرمج، ستستخدم ADO.NET بغض النظر عن نوع مصدر البيانات، ومن منظور كمصمم للتطبيق، عليك تحديد مزود NET Data Providers. والذي يلائم نوع مصدر البيانات الذي تستخدمه، كما عليك إرفاق كافة ملفات هذا المزود عند توزيع برنامجك. توجد العشرات من مزودات NET Data Providers. تمثل انواع مختلفة من مصادر البيانات، ويتم تحديث اصدارات كل فترات متعددة، عليك البحث دائما عن اخر اخبار المزود الذي تستخدمه في موقع الشركة المنتجة له. توجد ثلاثة انواع من مزودات البيانات مدعومة في اطار عمل NET Framework. هي:

:OLE DB .NET Data Provider

هذا النوع من المزودات يمكنك من استخدام مزودات OLE DB قديمة مبنية على تقنية COM. يفيدك كثيرا ان اردت استخدام نفس مصادر البيانات والمنجزة قبل تقنية NET. Microsoft، مع العلم ان استخدامها اقل كفاءة من استخدام مزودات NET. وذلك لإجراء عمليات تحويل الشيفرات من NET. إلى COM ومن COM إلى NET. (لا تقلق، فلست مسئولا عن هذه العمليات ان استخدمت مزودات خاصة بمصادر بيانات لـ Microsoft Access® مثلا).

:SQL Server .NET Data Provider

هذا المزود موجه -بشكل خاص- إلى مصادر البيانات المنجزة بالاصدار السابع من Microsoft SQL Server® وما بعده، وان اردت التعامل مع اصدارات اقدم، فعليك استخدام المزودات من النوع OLE DB .NET Data Provider السابقة.

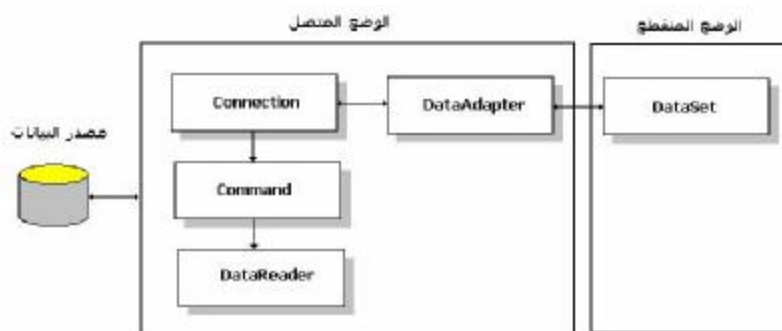
:ODBC .NET Data Provider

قبل تقنية NET. وقبل تقنيات الوصول إلى البيانات، كانت تقنية ODBC اول تقنية تمكن المبرمجين من الوصول إلى مصادر البيانات المختلفة عن طريق مشغلات ODBC Drivers المختلفة (الفكرة من المشغلات والمزودات شبيهة إلى حد كبير من ناحية نظرية). الغرض من المزود ODBC .NET Data Provides هو استخدام مشغلات ODBC Drivers للوصول إلى

مصادر البيانات (وهي اقل كفاءة حتى من مزودات .NET Data Providers OLE DB ولا انصحك بالاعتماد عليها).

فئات ADO.NET

تحتوي ADO.NET على مجموعة كبيرة من الفئات، ولكن اغلبها مشتقة من خمس فئات رئيسية هي: Connection، Command، DataAdapter، DataReader، و DataSet (شكل 17-2)، وعملية استيعاب هذه الفئات القاعدية سيسهل عليك الامر كثيرا للنزول إلى الفئات المشتقة وفهم طرق استخدامها والربط بين الكائنات المنشئة منها.



شكل 17-2: أبرز فئات ADO.NET.

سأفصل في جميع الفئات -السابق ذكرها- في هذا الفصل والفصل القادم بمشيئة الله، ولكن دعنا الآن نأخذ جولة سريعة حولها. حيث تكون البداية دائما وأبدا مع Connection، اذ يترتب عليك انشاء كائن من هذه الفئة عن طريقه تتصل بمصدر البيانات، كما ستلحق كل خصائص ومواصفات الاتصال (كاسم المستخدم، كلمة المرور، مزود .NET Data Provider المستخدم... الخ) مع هذا الكائن.

بعد انشاءك لكائن اتصال، سترسل مرجع لهذا الكائن إلى طرق وخصائص كائن اخر من النوع Command، يمكنك من تنفيذ جمل الاستعلام المختلفة (SELECT، DELETE، INSERT وغيرها) على مصدر البيانات والمحدد في الكائن Connection السابق.

بعد تنفيذ جمل الاستعلام في الكائن Command، ستكون النتائج متمثلة في سجلات Records تصل إليها بين ثنايا الكائن من النوع SqlDataReader، لتتمكن من قراءة كل سجل أو كل حقل على حدة، مع العلم أنك لن تستطيع تحديث مصدر البيانات من خلال هذا الكائن. الفئات الثلاث السابق ذكرها تستخدم في حالة الوضع المتصل Connected Mode، أما الكائنات من النوع DataSet فيوجد الكثير لأخبرك به حولها في الفصل القادم. وحتى ان نلتقي هناك، اعلم انها تمثل البيانات المأخوذة من مصدر البيانات. أخيراً، يمكنك الحصول على البيانات من مصدر البيانات أو إرسالها إليه، وإلحاقها إلى الكائنات من النوع DataSet عن طريق كائن من النوع DataAdapter، يمكنك اعتباره كهزمة الوصل التي تمكنك من ارسال/استقبال البيانات من وإلى كائنات Connection. في هذا الفصل سافصل في الفئات Command، Connection، و SqlDataReader. أما البقية فستأتي لاحقاً، وسأبدأ الآن بكائن الاتصال Connection.

كائن الاتصال Connection

في عالم ADO.NET، الاتصال الذي تجربيه مع مصدر بيانات هو كائن من النوع Connection. وقبل التعامل مع مصدر بيانات، عليك فتح اتصال معها (حتى ولو كنت في الوضع المنفصل Disconnected Mode). في هذا القسم من الفصل سنعرض الاساليب المتعددة للاتصال بمصادر البيانات.

إنشاء كائن اتصال

عندما تنوي إنشاء كائن اتصال Connection فانك لن تقوم بتعريفه من الفئة Connection السابقة، وإنما ستستخدم نوعين من الاتصال، النوع الاول للمزودات من النوع OLE DB .NET Data Provider حيث ستعرف كائن اتصال من الفئة OleDbConnection:

```
Dim msAccessCn As New OleDbConnection()
```

أما ان كان مصدر البيانات الذي تنوي الاتصال به يتبع للمزود SQL Server .NET Data Provider، فالفئة SqlConnection هي المسؤولة عن إنشاء كائنات من هذا النوع:

```
Dim msSQLCn As New SqlConnection()
```

كلا الفئتين تحتويان على الواجهة IDbConnection مما يمكنك من الاستفادة من مبدأ تعدد الواجهات Polymorphism وتتمكن -مثلا- من كتابة اجراء واحد يستقبل كلا النوعين لاختصار كتابة الشيفرات المكررة:

```
Sub ConnectionDB(ByVal dbCon As IDbConnection)
    ...
    ...
    ...
End Sub
```

نص الاتصال

بعد انشاء كائن الاتصال، عليك ارفاق نص الاتصال **Connection String** اليه، ونص الاتصال ما هو إلى قيمة حرفية (من النوع String) تحتوي على كل شيء تتعلق بعملية الاتصال بمصدر البيانات، كاسم مصدر البيانات، مسار قاعدة ملف قاعدة البيانات (أو قد تكون اسم الجهاز في الشبكة ان كنت معتمد على خادم قواعد بيانات كـ Microsoft SQL Server®)، اسم المستخدم، كلمة المرور... الخ، نص الاتصال التالي مناسب جدا لمصادر بيانات من النوع Microsoft Access®:

```
Dim connString As String = "Provider=Microsoft.Jet.OLEDB.4.0;" _
    & "Data Source=C:\Folder\Data.MDB;"
```

حيث Provider هو الاسم الكامل للمزود، و Data Source اسم ملف قاعدة البيانات. المزيد ايضا، يمكنك اسناد وقت الانتظار **Connection Timeout** بين ثانيا نص الاتصال، ووقت الانتظار هي اطول فترة التي سيضل الكائن منتظرا ردة الفعل من مصدر البيانات عند الاتصال، فنص الاتصال التالي سينتظر 10 ثواني لفتح الاتصال مع مصدر البيانات، وان انتهت الفترة ولم تحدث أي ردة فعل من مصدر البيانات نفسه، سيظهر خطأ (القيمة الافتراضية هي 15 ثانية):

```
Dim connString As String = "Provider=Microsoft.Jet.OLEDB.4.0;" _
    & "Data Source=C:\Folder\Data.MDB; Connection Timeout=10"
```

بعد كتابتك لنص اتصال، أسنده فورا إلى الخاصية **ConnectionString** التابعة لكائن الاتصال، كما يمكنك ارسال نص الاتصال مباشرة إلى مشيد الفئة لحظة انشاء الكائن:

```
' اسناد قيمة للخاصية
Dim msAccessCn As New OleDbConnection()

msAccessCn.ConnectionString = connString
```

```
ارسالها مع المشيد '
Dim msAccessCn As New OleDbConnection(connString)
```

ملاحظة

بدلاً من كتابة نص الاتصال كاملاً في كل مرة تنوي إنشاء كائن اتصال جديد، يفضل وضعه في متغير أو ثابت عام على مستوى المشروع، أما أفضل نصيحة أسطرها لك فهي حفظ نصوص الاتصال في ملف الإعدادات project.exe.config أو أي ملف خارجي آخر، وذلك لتمكين المستخدمين من تغيير نص الاتصال في حال ما أن تم تغيير موقع مصدر البيانات.

يمكنك الاستعلام عن معظم القيم التي اسندتها إلى نص الاتصال عن طريق مجموعة من الخصائص للقراءة فقط `ReadOnly`، كالخاصية:

```
Dim msAccessCn As New OleDbConnection(connString)

MsgBox(msAccessCn.Provider) ' Microsoft.Jet.OLEDB.4.0
MsgBox(msAccessCn.DataSource) ' C:\Folder\Data.MDB
MsgBox(msAccessCn.ConnectionTimeout) ' 10
```

عندما تتعامل مع الكائنات من النوع `SqlConnection`، فعليك تجاهل اسم المزود عند كتابتك لنص الاتصال، والسبب قد يبدو بديهياً أن علمت أن الكائنات من النوع `SqlConnection` لا تقبل إلا المزودات من النوع `SQL Server .NET Data Provider`:

```
Dim msSQLCn As New SqlConnection()

msSQLCn.ConnectionString = "Data Source=DEV4ARABS_SERVER;" _
& "User ID=تركى العسيري; Password=تلم فيها;" _
& "Initial Catalog=قاعدة بيانات المقالات"
```

فتح وإغلاق الاتصالات

بعد إسنادك لنص الاتصال المناسب للخاصية `ConnectionString`، يمكنك البدء بفتح الاتصال مع مصدر البيانات باستدعاء الطريقة `Open()`:

```
Dim msAccessCn As New OleDbConnection(connString)
Dim msSQLCn As New SqlConnection(connString2)

msAccessCn.Open ()
msSQLCn.Open ()
```

ملاحظة

المتغير connString السابق اقصد فيه نص اتصال Connection String، لذلك في كل مرة تجده في هذا الفصل والفصول اللاحقة لا تستغرب من وجوده.

ومن الضروري جدا اغلاق الاتصال عند عدم الحاجة اليه باستدعاء الطريقة Close():

```
msAccessCn.Close ()
msSQLCn.Close ()
```

تستطيع معرفة حالة الاتصال عن طريق الخاصية State التابعة لكائن الاتصال، والتي قد تكون قيمة أو أكثر من القيم: Closed الاتصال مغلق، Open الاتصال مفتوح، Connecting جاري فتح الاتصال، Executing يتم تنفيذ امر استعلام على الاتصال، و Fetching جاري الحصول على بيانات من سجلات مصدر البيانات:

```
Dim cn As New OleDbConnection()
...
...
...

If (cn.State And ConnectionState.Open) <> 0 Then
    cn.Close()
End If
```

المزيد ايضا، عند تغيير حالة الاتصال من Open إلى Closed (أو العكس) سيتم تفجير الحدث StateChange والخاص بكائن الاتصال:

```
Dim WithEvents cn As New OleDbConnection()

Sub cn_StateChange(ByVal sender As Object, _
    ByVal e As System.Data.StateChangeEventArgs) Handles _
    cn.StateChange
```

```

If (e.CurrentState And ConnectionState.Open) <> 0 Then
    Label1.Text = "تم فتح الاتصال"
Elseif e.CurrentState = ConnectionState.Closed Then
    Label1.Text = "تم إغلاق الاتصال"
End If
End Sub

```

تفادي الاستثناءات:

عند التعامل مع مصادر البيانات، فإن نسبة وقوع الاستثناءات كبيرة جداً لأي سبب أو خلل فني، لذلك ينصح بشدة من تفادي الاستثناءات ووضع الشيفرات الخاصة بمصادر البيانات داخل التركيب
:Try ... End Try

```

Dim cn As New OleDbConnection()

Try
    cn.Open()
    ...
    ...
Catch ex As Exception
    MsgBox(ex.Message)
End Try

```

بالنسبة للطريقة Close() السابقة، عليك استدعاؤها دائماً لتغلق الاتصال، حيث إن الاتصالات لا يتم إغلاقها إلا لحظة الموت الحقيقي للكائنات (عندما تبدأ المجموعة Garbage Collection عملها)، وبما أننا لا نعلم متى سيحدث هذا فعلينا استدعاؤها دائماً، سواء وقع استثناء أو لم يقع:

```

Dim cn As New OleDbConnection()

Try
    cn.Open()
    ...
    ...
Catch ex As Exception
    MsgBox(ex.Message)
Finally
    cn.Close()
End Try

```

كما يفضل الاعتماد على كائنات الاستثناءات OleDbException أو SQLException للتفريق بين الاستثناءات الخاصة بكائنات الاتصالات والاستثناءات الأخرى:

```
Try
...
...
Catch ex As OleDbException
...
...
Catch ex As Exception
...
...
End Try
```

انظر ايضا

لمزيد من التفاصيل حول الاستثناءات Exceptions وطرق تفاديها، راجع الفصل السابع **اكتشاف الأخطاء**.

كائن الأوامر Command

بعد تكوين الاتصال مع قاعدة البيانات، تأتي الخطوة التالية وهي إرسال جمل الاستعلام SQL إلى قاعدة البيانات لتعديل محتوياتها، في هذا القسم سنرى كيف يمكنك الاستفادة من كائن الاتصال Connection وتعديل بيانات قاعدة البيانات عن طريق كائن الاوامر Command.

انظر ايضا

سأستخدم في هذا القسم -والقسم الذي يليه- مجموعة من جمل الاستعلام SQL، وسأفترض بان لديك خلفية في لغة الاستعلام SQL أو انك قد قرأت المحلق ب **لغة الاستعلام** SQL والملقى في نهاية هذا الكتاب.

إنشاء كائن أوامر

عندما نتوي إنشاء كائن أوامر Command فانك لن تقوم بتعريفه من الفئة Command، وإنما ستستخدم نوع يماثل مزود كائن الاتصال Connection، فإن كان مزود كائن الاتصال من النوع OLE DB .NET Data Provider، ستعرف كائن أوامر من الفئة OleDbCommand:

```
Dim msAccessCmd As New OleDbCommand()
```

اما ان كان مزود كائن الاتصال الذي نتوي استخدامه يتبع للمزود SQL Server .NET Data Provider، فالفئة SqlCommand هي المسؤولة عن إنشاء كائنات اوامر خاصة لها:

```
Dim msSQLCmb As New SqlCommand()
```

كلا الفئتين تحتويان على الواجهة IDbCommand مما يمكنك من الاستفادة من مبدأ تعدد الواجهات Polymorphism وتتمكن -مثلا- من كتابة إجراء واحد يستقبل كلا النوعين للتقليص من عدد الشيفرات المكررة:

```
Sub CommandDB(ByVal dbCmd As IDbCommand)
    ...
    ...
    ...
End Sub
```

الربط مع اتصال

بعد إنشائك لكائن أوامر Command، اول خطوة عليك إنجازها هي ربطه مع كائن اتصال Connection، يمكنك اسناد مرجع إلى كائن اتصال عن طريق الخاصية Connection:

```
Dim cn As New OleDbConnection(connString)
Dim cmd As New OleDbCommand()

cn.Open()

cmd.Connection = cn
```

ولا تنسى ضرورة توافق نوع كائن الاوامر مع نوع كائن الاتصال، فلو كان كائن الاتصال يتبع مزود من النوع SQL .NET Data Provider فقد تم إنشائه من الفئة SqlConnection، لذلك

عليك الاعتماد على الفئة SqlCommand عوضا عن OleDbCommand للربط الصحيح مع الاتصال:

```
Dim cn As New SqlConnection(connString)
Dim cmd As New SqlCommand()

cn.Open()

cmd.Connection = cn
```

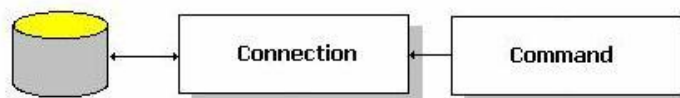
ضع في عين الاعتبار، ان كائن الاوامر Command مرتبط ارتباطا وثيقا بكائن الاتصال Connection، فلو تم -مثلا- اغلاق كائن الاتصال بالطريقة Close()، فلن تتمكن من تنفيذ جمل الاستعلام SQL مع كائن الاوامر:

```
Dim cn As New OleDbConnection(connString)
Dim cmd As New OleDbCommand()

cn.Open()
cmd.Connection = cn
cn.Close()
' لن تتمكن من الاستفادة من كائن الاوامر '
cmd.ExecuteNonQuery()
```

سيظهر لك السبب واضحا ان علمت ان كائن الاوامر لا يصل إلى مصدر البيانات بشكل مباشر، وانما يعتمد على كائن اتصال والذي بدوره يصل إلى مصدر بيانات (شكل 17-3).

مصدر البيانات



شكل 17-3: الكائن Command يعتمد على الكائن Connection للوصول إلى مصدر بيانات.

تنفيذ جمل الاستعلام SQL

ان اردت تنفيذ جمل استعلام على الكائن Command، فعليك تحديد نوع جملة الاستعلام SQL التي تود تنفيذها، هل هي جملة استعلامية تقليدية أو جملة تنفيذية؟
 الجمل الاستعلامية هي تلك الجمل التي لا تؤثر على سجلات قاعدة البيانات وانما تقوم بقراءة محتوياتها، في لغة الاستعلام SQL نستخدم الامر SELECT لهذا النوع من الجمل، وسترى في الفقرة التالية قراءة السجلات كيف يمكنك تنفيذها على كائن Command باستدعاء الطرق ExecuteXXX().

اما ان كانت جمل الاستعلام SQL هي جمل تنفيذية، عليك استخدام الطريقة ExecuteNonQuery() والتي تعود بقيمة عددية تمثل عدد السجلات التي تأثرت. الجمل التنفيذية -التي اقصدها في هذا السياق- هي تلك الجمل التي تحدث تغييرا في سجلات جداول قاعدة البيانات وهي اما تكون UPDATE، INSERT INTO، أو DELETE.
 حتى تتمكن من استدعاء الطريقة ExecuteNonQuery()، عليك اولا كتابة جملة الاستعلام SQL في الخاصية CommandText والتابعة للكائن Command:

```
Dim cn As New OleDbConnection(connString)
cn.Open()

Dim sqlStatement As String = "UPDATE [جدول الرواتب] SET [الراتب] = " & " = 5000 WHERE [رقم الموظف] = 99999"
Dim cmd As New OleDbCommand()

cmd.Connection = cn
cmd.CommandText = sqlStatement
cmd.ExecuteNonQuery()
```

يمكنك دمج السطرين والمتعلقين بتعديل قيم الخاصيتين Connection و CommandText في سطر واحد، وذلك بارسال امر جملة الاستعلام وكائن الاتصال مع مشيد الفئة Command:

```
Dim cmd As New OleDbCommand(sqlStatement, cn)
```

اخيرا، دعني اعيد تذكيرك بضرورة تفادي الاستثناءات لحظة تنفيذ جمل الاستعلام -فد تتغير صلاحياتك على مصدر البيانات أو يحدث أي خلل فني في عملية التحديث:

```
Try
    cmd.ExecuteNonQuery ()
    ...
    ...
Catch ex As Exception
    ...
    ...
End Try
```

قراءة السجلات

عندما نتوي استخدام الامر SELECT في جمل الاستعلام لقراءة السجلات، فيمكن اختيار طريقة من ثلاث طرق تابعة للكائن Command هي: ExecuteReader()، ExecuteScalar()، و ExecuteXMLReader.

الطريقة ExecuteReader():

الطريقة ExecuteReader() تعود بكائن من النوع DataReader تمثل نتيجة جملة الاستعلام في الكائن Command (سأحدث عن الكائن DataReader لاحقاً في القسم كائن البيانات DataReader من هذا الفصل):

```
Dim cmd As New OleDbCommand("SELECT * FROM [جدول الخسنوات]", cn)
Dim dr As OleDbDataReader = cmd.ExecuteReader()

Do While dr.Read()
    ListBox1.Items.Add(CInt(dr.Item("الوزن")))
Loop

dr.Close()
```

الطريقة ExecuteScalar():

تستخدم الطريقة ExecuteScalar() لقراءة **حقل Field** واحد فقط من حقول السجل، وتعود بقيمة تمثل ذلك الحقل:

```
Dim cmd As New OleDbCommand("SELECT [اسم الضفدع] FROM " _
    & " [الضفادع البشرية] WHERE [المعرف] = 1", cn)

Dim Name As String = cmd.ExecuteScalar()
```

تدليك الطريقة ExecuteScalar() كثيرا ان اردت قراءة قيمة واحدة من من السجل وذلك لزيادة سرعة التنفيذ، ان كانت جملة الاستعلام تعود بأكثر من حقل، فالحقل الاول هو الذي سيتم قراءته، وان كانت جملة الاستعلام تعود بأكثر من سجل، فسيتم قراءة الحقل التابع للسجل الاول.

ملاحظة

لا اقصد -في فصول الجزء الرابع برمجة قواعد البيانات بالتحديد- من كلمة الحقل Field الذي تعرفه في الفئات، وانما اقصد به العامود الموجود في احد جداول قاعدة البيانات.

يمكنك الاستفادة من الطريقة ExecuteScalar() في معرفة عدد السجلات بشكل سريع - على سبيل المثال لا الحصر:

```
Dim cmd As New OleDbCommand("SELECT COUNT(*) FROM " _
    & "[الصفادع البشرية]", cn)

Dim countOfFrogs As Integer = CInt(cmd.ExecuteScalar())
```

الطريقة ExecuteXMLReader():

ان استخدم المزود SQL Server .NET Data Provider، فستتمكن من استدعاء الطريقة ExecuteXMLReader()، حيث ان قواعد البيانات المعتمدة على انظمة Microsoft SQL Server®، يمكن استخدام الامر FOR XML من اوامر لغة الاستعلام SQL معها، والتي تعود بالسجلات بهيئة XML.

اذا كانت الطريقة ExecuteReader() تعود بكائن من النوع SqlDataReader، فان الطريقة ExecuteXMLReader() تعود بكائن من النوع System.Xml.XmlReader، واستخدامها شبيه -إلى حد كبير- مع SqlDataReader:

```
Dim cmd As New SqlCommand("SELECT * FROM [جدول الذكريات]", cn)
Dim xmlr As System.Xml.XmlReader = cmd.ExecuteXmlReader()

Do While xmlr.Read()
    MsgBox(xmlr.Value)
Loop

xmlr.Close()
```

انظر ايضا

سأعود للطريقة ExecuteXmlReader() وكيفية استخدامها مع الفئة XmlReader في الفصل التاسع عشر تكامل ADO.NET و XML.

كائن البيانات DataReader

بعد تنفيذ جملة الاستعلام في الكائن Command باستدعاء الطريقة السابقة ExecuteReader()، ستعود هذه الطريقة بكائن بيانات من النوع DataReader يمثل جميع السجلات الناتجة من جملة الاستعلام، وقبل ان نرى كيف يمكنك الاستفادة من هذا الكائن، من الجيد معرفة كيف يمكنك الحصول عليه بأي إنشائه.

إنشاء كائن بيانات

عندما نتوي إنشاء كائن بيانات DataReader فانك لن تستطيع استخدام الامر New، وانما ستضطر إلى استخدام الطريقة ExecuteReader() والتابعة للكائن Command، فان كان مزود الكائن Command من النوع .NET Data Provider OLE DB، فانك ستعرف كائن بيانات من الفئة OleDbDataReader:

```
Dim cmd As New OleDbCommand("SELECT * FROM [جدول]", cn)
Dim dr As OleDbDataReader = cmd.ExecuteReader()
```

اما ان كان مزود كائن الاوامر Command الذي تتوي استخدامه يتبع للمزود SQL Server .NET Data Provider، فالفئة SqlDataReader هي المسئولة عن إنشاء كائنات بيانات خاصة ملائمة ومتوافقة معها:

```
Dim cmd As New SqlCommand("SELECT * FROM [جدول]", cn)
Dim dr As SqlDataReader = cmd.ExecuteReader
```

مرة اخرى، الواجهة IDataReader مشمولة في كلا الفئتين OleDbDataReader و SqlDataReader، فيمكنك تعريف اجراء يستقبل في وسيطته كائن من كلا النوعين:

```
Sub ReaderDB(ByVal dr As IDataReader)
    ...
    ...
    ...
End Sub
```

قراءة السجلات

بعد إنشائك لكائن بيانات `DataReader`، تستطيع البدء في قراءة سجلاته في خطوتين، الأولى باستدعاء الطريقة `Read()` لتحميل حقول السجل، والثانية تتم فيها قراءة قيمة الخاصية `Item` والتي ترسل معها اسم الحقل المراد قراءته:

```
Dim dr As OleDbDataReader = cmd.ExecuteReader()

dr.Read()
MsgBox ( dr.Item("الاسم") & " - " & CInt(dr.Item("العمر")) )

dr.Read()
MsgBox ( dr.Item("الاسم") & " - " & CInt(dr.Item("العمر")) )

dr.Close()
```

كما يمكنك استخدام رقم فهرس الحقل بدلا من كتابة اسمه:

```
MsgBox ( dr(0) & " - " & CInt(dr(1)) )
```

ملاحظة

عند استخدام رقم فهرس الحقل كما في الشيفرة الأخيرة، عليك الانتباه أن الترقيم يعتمد على مكان وجود الحقل في جملة الاستعلام وليس ترتيب الحقل في نفس جدول قاعدة البيانات الأصلي، لذلك قد يختلف رقم فهرس الحقل من جملة استعلام إلى أخرى.

في كل مرة تستدعي فيها الطريقة `Read()` سيتم نقل المؤشر إلى السجل التالي، وستعود الطريقة بالقيمة `False` ان وصلت إلى نهاية السجلات، لذلك الاستخدام الأمثل لها يكون في حلقة `Do ... Loop` بهذا الشكل:

```
Dim dr As OleDbDataReader = cmd.ExecuteReader()
...
Do While dr.Read()
    MsgBox (dr.Item("الاسم") & " - " & CInt(dr.Item("العمر")))
Loop

dr.Close()
```

ملاحظة

Item هي الخاصية الافتراضية لكائن البيانات DataReader، لذلك يمكنك اختصار استدعائه بهذا الشكل:

```
MsgBox ( dr("الاسم") & " - " & CInt(dr("العمر")) )
```

المزيد ايضا، الخاصية Item تعود دائما بقيمة من النوع Object، مما يضطرك إلى الاعتماد على دوال التحويل (CInt()، CLng()، CSng()... الخ) لقراءة القيمة، مع ذلك يمكنك الاعتماد على الطرق (Getxxx()) التي تعود بالنوع المكافئ -دون الحاجة لاستخدام دوال التحويل:

```
Dim age As Integer

age = CInt(dr.Item("العمر"))
age = dr.GetInt32(1)
```

من المهم التنبيه هنا بضرورة اغلاق كائن DataReader باستدعاء طريقته Close()، السبب في ذلك ليس فقط من اجل تحرير مصادر النظام، وانما يتعدى ذلك بكثير، اذ بمجرد قيامك بإنشاء كائن DataReader، سيتم شغل كافة العمليات الاخرى على كائن الاتصال Connection والوامر Command، ولن تتمكن من عمل أي شيء الا استدعاء الطريقة Close() لكائن الاتصال في هذه الحالة.

اخيرا، تحتوي الكائنات من النوع DataReader على مجموعة اضافية من الطرق والخصائص المفيدة كالخاصية GetName التي تعود باسم الحقل (وليس قيمته)، الخاصية FieldCounter التي تعود بعدد الحقول، والاهم من ذلك الطريقة IsDBNull التي تعود بالقيمة True ان كان الحقل فارغ Null (عليك التحقق منها دائما قبل قراءة قيمة الحقل حتى تتفادى ظهور الاستثناءات).

خاص بمستخدمي Microsoft SQL Server ®

عند التعامل مع مزودات من النوع .NET Data Provider SQL Server، يفضل دائما قراءة قيمة الحقل باستخدام الطرق (GetSqlxxx) عوضا عن الخاصية Item:

```
Dim dr As SqlDataReader = cmd.ExecuteReader()  
dr.GetSqlInt32(0)
```

تتصح مستندات .NET Documentation. باستدعاء الطرق السابقة لاتقاء شر التحويلات الغير دقيقة (التضييق Narrowing)، كما انها ستؤدي إلى سرعة وزيادة في كفاءة التنفيذ، ولا تنسى ان حقول جداول Microsoft SQL Server® تحتوي على أنواع لن تجد مكافئ لها في إطار عمل .NET Framework. فالنوع SqlDecimal حجمه 38 بت، بينما النوع المقابل له في إطار عمل .NET Framework حجمه لا يتجاوز 28 بت وهو المعروف بـ Decimal.

ملاحظة

الانواع التي تعود بها الطرق (GetSqlxxx) معرفة في مجال الاسماء System.Data.SqlTypes وليس System.

الجدول الظاهر في الصفحة المقابلة يعرض لك هذه الطرق والنوع الذي تعود به في مجال الاسماء System.Data.SqlTypes لها، كما يشمل العمود الاخير النوع المكافئ لتعريف الحقول في جداول الخادم Microsoft SQL Server®:

الطريقة	النوع الذي تعود به	المكافئ له في خادم SQL Server®
GetSqlBoolean()	SqlBoolean	bit
GetSqlByte()	SqlByte	tinyint
GetSqlInt16()	SqlInt16	smallint
GetSqlInt32()	SqlInt32	int
GetSqlInt64()	SqlInt64	bigint
GetSqlSingle()	SqlSingle	real
GetSqlDouble()	SqlDouble	float
GetSqlDecimal()	SqlDecimal	decimal
GetSqlDateTime()	SqlDateTime	datetime
		smalldatetime
GetSqlMoney()	SqlMoney	money
		smallmoney
GetSqlString()	SqlString	char
		nchar
		ntext
		nvarchar
		sysname
		text
		varchar
GetSqlBinary()	SqlBinary	binary
		varbinary
		image
		timestamp
GetSqlObject()	SqlObject	sql_variant

قراءة نتائج متعددة

بعض انواع مصادر البيانات (كخادم Microsoft SQL Server ®) تسمح لك بتنفيذ جملة استعلام في وقت واحد، وذلك بفصل جمل الاستعلام بالفاصلة المنقوطة ";":

```
SELECT * FROM [جدول الحسنات] WHERE [الوزن] < 50 ;
SELECT * FROM [جدول الحسنات] WHERE [الوزن] > 90
```

تنفيذ اكثر من جملة استعلام في امر واحد يزيد من سرعة تنفيذ جملة الاستعلام باضعاف اضعاف المرات، وذلك لان عملية المسح على سجلات الجدول ستكون واحدة وليس اثنتين. عند التعامل مع جمل الاستعلامات المتعددة، استدعي الطريقة NextResult() ان أردت قراءة نتائج جملة الاستعلام الاخرى والتي ستعود بالقيمة False ان لم توجد أي جملة استعلام اخرى:

```

Dim cn As New SqlConnection(connString)
cn.Open()
Dim cmd As New SqlCommand("SELECT * FROM [جدول الحسنات] " _
    & "WHERE [الوزن] < 50; SELECT * FROM [جدول الحسنات] " _
    & "WHERE [الوزن] > 90", cn)

Dim dr As SqlDataReader = cmd.ExecuteReader()

Do
    Do While dr.Read
        ListBox1.Items.Add(dr.Item("الاسم"))
    Loop
Loop While dr.NextResult

dr.Close()
cn.Close()

```

كان هذا الفصل مدخلا لك لبرمجة قواعد البيانات باستخدام ADO.NET في الوضع المتصل Connected Mode، وبقي عرض الجزء الهام من ADO.NET والذي يستخدم في سيناريو الوضع المنفصل Disconnected Mode كما سترى بالفصل التالي. على فكرة، نسيت إخبارك بأن اختصار ADO.NET هو ActiveX Data Objects وهو اسم مستحدث من تقنية ADO القديمة والمبنية على تقنية COM الأقدم منها في بنيتها التحتية -الشغلة شغلة تسويق يا عيال!

!Microsoft

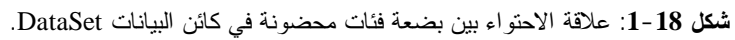
ADO.NET للوضع المنفصل

اسلوب الوضع المنفصل يوعد بكفاءة عالية عند الحديث عن تطبيقات العميل/الخادم -Client- Server Applications، فمشاكل الازدحام والضغط الشديد على خوادم قواعد البيانات لن تحدث بعد الآن، اذ ان عملية استخلاص البيانات ووضعها في أجهزة العملاء، سيتم معالجتها والتعامل معها في مصادر النظام الخاصة بكل عميل من العملاء، مما يخفف الضغط على الخادم. في هذا الفصل سنرى كيف تعمل ADO.NET للاتصال المنفصل، وقد قسمته إلى قسمين رئيسيين الاول يعرض لك الفئة DataSet لمعالجة البيانات، والثاني DataAdapter التي تمثل الجسر الذي سترسل به بياناتك بعد معالجتها.

كائن البيانات DataSet

في سياق الوضع المنفصل، فان كائن البيانات هو DataSet وليس DataReader كما رأينا في الفصل السابق. الكائن DataSet هو كائن حاضن لعشرات الكائنات والمجموعات يمكنك مراجعة كافة تفاصيلها في مستندات .NET Documentation. حيث يعرض لك (الشكل 18-1 بالصفحة التالية) بضعة فئات منها والتي يمكنك ليس فقط من تحرير السجلات، بل اعادة هيكلتها في تغيير حقولها او تعديل العلاقات بينها.

دعني انوه هنا بان (شكل 18-1) يبين لك علاقات الاحتواء Containment بين الفئات وليس الوراثة والاشتقاق الوراثي، وطريقة الوصول إلى الفئات المحبونة من الكائن الرئيسي تكون دائما عن طريق الخاصية Item لكل مجموعة من المجموعات. سترى كيف يتم ذلك بعد قراءتك لل فقرات التالية.



كما ترى في (شكل 18-1)، كائن البيانات يحتوي على مجموعتين رئيسيتين هما DataTableCollection و DataRelationCollection، الأولى تصل إليها عن طريق الخاصية Tables والثانية من الخاصية Relations:

```
Dim ds As New DataSet()  
...  
...  
MsgBox(ds.Tables.Count)  
MsgBox(ds.Relations.Count)
```

وكما ذكرت قبل قليل، إن اردت الوصول إلى احد كائنات المجموعة، فاستخدم دائما الخاصية Item والتابعة للمجموعة:

```
Dim ds As New DataSet()  
Dim dt As DataTable  
  
dt = ds.Tables.Item("جدول العملاء")
```

كما يمكنك استخدام حلقة For ... Each او الطرق Add()، Insert()، Remove()... الخ فهي مجموعة يا عزيزي:

```
Dim ds As New DataSet()  
Dim dt As DataTable  
  
For Each dt In ds.Tables  
    MsgBox(dt.TableName)  
Next  
  
Dim ds As New DataSet()  
Dim dt As New DataTable()  
...  
...  
ds.Tables.Add(dt)
```

الوصول إلى الكائنات المحفوظة يتم من خلال الكائن الحاضن لها، او الكائن DataSet الرئيسي بشكل مباشر:

```
Dim ds As New DataSet()
Dim dc As DataColumn

For Each dc In ds.Tables.Item("جدول العملاء").Columns
    MsgBox(dc.ColumnName)
Next
```

أتمنى ان تكون قد اتضحت لك فكرة التعامل مع هذا الهرم الكائني، وان كانت لم تتضح بعد فاعتقد انك بحاجة إلى العودة إلى الفصل السادس الفئات الأساسية ومعرفة كيف التعامل مع المجموعات Collection (الفئات التي تحتوي على الواجهة ICollection)، اما ان كنت مدرك تماما للوضع فاستعن بالله وابدأ بالفئة DataTable.

الفئة DataTable

الفئة DataTable تمثل جدول من جداول قاعدة البيانات، تحتوي على مجموعة كبيرة من الطرق والخصائص كالخاصية Rows (وهي مجموعة من النوع DataRowCollection) التي تمثل السجلات، والخاصية Columns (مجموعة من النوع DataColumnCollection) تمثل أعمدة (حقول) الجدول.

سأتحدث عن الخاصيتين السابقتين في الفقرتين التاليتين، ودعني هنا اذكر لك الخاصية TableName والتي تمثل اسم الجدول والتي تستطيع الاستغناء عنها بارسال اسم الجدول كمشيد للفئة لحظة انشاء الكائن:

```
Dim Table1 As New DataTable()
Table1.TableName = "جدول2"

Dim Table2 As New DataTable("جدول2")
```

يمكنك تعريف المفتاح الابتدائي Primary key لكل جدول عن طريق اسناد مصفوفة من النوع DataColumn إلى الخاصية PrimaryKey، والسبب في كون القيمة التي تستقبلها الخاصية PrimaryKey هي مصفوفة يتضح إن علمت أن المفتاح الابتدائي يمكن ان يشمل اكثر من حقل:

```
Dim pkeys() As DataColumn = {New DataColumn(), New DataColumn()}
Dim myTable As New DataTable()

myTable.PrimaryKey = pkeys
```

اخيرا، مجموعة من الأحداث التابعة لهذه الفئة كالحديث ColumnChanging والذي يتم تنفيذه عندما يتم اضافة الحقول ثم يليه الحدث ColumnChanged، كذلك الحال مع الحدث RowChanging اضافة سجل جديد ليليه الحدث RowChanged، اما الاحداث RowDeleted و RowDeleting فيرتبطان بعمليات حذف السجلات.

الفئة DataRow

الفئة DataRow تمثل سجل كامل من سجلات الجدول، لن تتمكن من انشاء كائن باستخدام New من الفئة DataRow، وانما يتحتم عليك إنشائه بطريقتين، الاول باستخدام الطريقة NewRaw() التابعة للفئة:

```
Dim table As New DataTable()  
...  
...  
Dim dr As DataRow = Employees.NewRow()
```

والثانية تعريف مصفوفة من النوع Object تمثل قيم الحقول في السجل، ومن ثم تضيفها إلى الخاصية Rows و التابعة للكائن DataTable:

```
Dim dr2 As Object() = {"محمد عبدالله", 40, True}  
table.Rows.Add(dr2)
```

ان عرفت كائن بالطريقة الاولى، فيمكنك اسناد قيم الحقول عن طريق الخاصية Item، كما يمكن ايضا قراءة الحقول من نفس الخاصية ان استخدمت الطريقة الثانية:

```
dr.Item("الاسم") = "محمد عبدالله"  
dr.Item("العمر") = 40  
dr.Item("متزوج") = True  
  
Dim x As String = tables.Rows(0).Item("الاسم")  
...  
...
```

ملاحظة

الخاصية Item هي الخاصية الافتراضية للكائن DataRow، لذلك يمكنك تجاهلها ان اردت:

```
dr("الاسم") = "محمد عبدالله"
dr("العمر") = 40
dr("متزوج") = True
```

الفئة DataColumn

الفئة DataColumn تمثل حقل من حقول الجدول، تحتوي على مجموعة من الخصائص ابرزها خاصية اسم الحقل ColumnName والتي يمكنك ارسالها كمثبيد:

```
Dim nameField As New DataColumn()
nameField.ColumnName = "حقل1"

Dim nameField2 As New DataColumn("حقل2")
```

خصائص تفيدك للحقول التي تنوي ان تجعلها كحقول معرفة، الخاصية AutoIncrement والتي تسند لها القيمة True ليتم زيادة قيمة الحقل مع كل سجل جديد بمقدار العدد الذي تضعه في الخاصية AutoIncrementSeed، وهناك ايضا الخاصيتان AllowDBNull و Unique اسند القيمة False إلى الاولى ان اردت منع القيم الخالية Null لهذا الحقل، واسند القيمة True إلى الثانية ان اردت عدم تكرار قيمة الحقل في اكثر من سجل:

```
Dim idField As New DataColumn()
With idField
    .AutoIncrement = True
    .AutoIncrementSeed = 1
    .AllowDBNull = False
    .Unique = True
End With
```

الفئة DataRelation

العلاقات Relationships بين الجداول يمكنك إنشائها عن طريقة الفئة DataRelation، اسند كائن الجدول الأساسي في الخاصية ParentColumns، وكائن الجدول التابع في الخاصية ChildColumns، ولا تنسى تعريف اسم العلاقة في الخاصية RelationNames :

```
Dim Table1 As New DataTable()
Dim Table2 As New DataTable()
...
...
Dim rel As New Relation

rel.RelationName = "علاقة1"
rel.ParentColumns(Table1.Columns("حقل المفتاح الابتدائي"))
rel.ChildColumns(Table2.Columns("حقل المفتاح الاجني"))
```

كما يمكنك اختصار الاسنادات الثلاث الاخيرة عن طريق مشيد الفئة:

```
Dim rel As New DataRelation("علاقة1", _
    Table1.Columns("حقل المفتاح الابتدائي"), _
    Table2.Columns("حقل المفتاح الاجني"))
```

ملاحظة

لا بد ان يكون كلا الجدولين معرفان في نفس كائن البيانات DataSet حتى تتمكن من تكوين علاقة بينهما.

إنشاء كائن بيانات DataSet من الصفر

في الحقيقة، لن تحتاج إلى انشاء كائن بيانات DataSet بنفسك، اذ انك ستحصل على كائن بيانات جاهز من مصدر بيانات وتقوم بتعديله كما سترى في القسم التالي من هذا الفصل **الفئة DataAdapter**. مع ذلك، سأريك كيف يمكنك الربط بين كل ما تعلمته في الفقرات السابقة لانشاء كائن بيانات DataSet من الصفر.

في الخطوات التالية سنحاول تصميم جدولين (شكل 18-2) تربطهما علاقة 1-ن بين رقم المعرف للموظف، والعمليات التي قام بها.

المبيعات			الموظفين	
رقم الموظف	دفع نقدي	المبلغ	المعرف	الاسم
1	نعم	ر.س. 100.00	1	عباس السريج
1	نعم	ر.س. 200.00	2	برعي ابو جبهة
3	لا	ر.س. 150.00	3	عبود اللوح
2	نعم	ر.س. 120.00		

شكل 18-2: كائن بيانات DataSet يمثل جدولان قمنا بإنشائهما من الصفر.

الخطوة الاولى هي اسهل الخطوات والتي تتطلب منك فقط انشاء كائن من النوع DataSet:



```
Dim myDataSet As New DataSet()
```

قد تحتاج إلى إرسال اسم لكائن البيانات مع مشيد الفئة -امر اختياري:



```
Dim myDataSetAs New DataSet("المبيعات")
```

بعد ذلك، نبدأ بتصميم الجداول، أنشئ كائنين من النوع DataTable وارسل اسماء الجداول

مع مشيداته -ايضا امر اختياري:



```
Dim Employees As New DataTable("الموظفين")
Dim Sales As New DataTable("المبيعات")
```

ان كنت تتوي باضافة الحقول، فيمكن تعريف كائنات من النوع DataColumn، ولكني

أفضل إضافتها بشكل مباشر مع الطريقة Add() والخاصة بالمجموعة Columns لكل جدول:



```
With Employees.Columns.Add("المعرف", GetType(Integer))
    .AutoIncrement = True
    .AutoIncrementSeed = 1
    .AllowDBNull = False
    .Unique = True
End With
With Employees.Columns.Add("الاسم", GetType(String))
    .MaxLength = 100
End With

Sales.Columns.Add("رقم الموظف", GetType(Integer))
Sales.Columns.Add("دفع نقدي", GetType(Boolean))
Sales.Columns.Add("المبلغ", GetType(Decimal))
```

بالنسبة للحقل الاول والخاص بالجدول الاول، فاغلب الظن ان يكون هو المفتاح الابتدائي

Primary Key، لذلك عليك اسناد مصفوفة من النوع DataColumn إلى الخاصية

:PrimaryKey



```
Employees.PrimaryKey = New DataColumn() {Employees.Columns("المعرف")}
```

بعد الانتهاء من تصميم حقول الجداول وتعريف المفاتيح المبدئية Primary keys، حان الوقت لإنشاء علاقة 1-ن تربط حقل **المعرف** من الجدول الاول إلى حقل **رقم الموظف** في الجدول الثاني، سنعرف كائن من النوع DataRelation بكل تأكيد:



```
Dim rel As New DataRelation("علاقة1", _
    Employees.Columns("المعرف"), _
    Sales.Columns("رقم الموظف"))
```

اخيرا، اهم خطوة هي الخطوة الاخيرة وهي اضافة كلا الجدولين إلى كائن البيانات DataSet، ولاتنسى اضافة العلاقة التي انجزها للتو:



```
myDataSet.Tables.Add(Employees)
myDataSet.Tables.Add(Sales)
myDataSet.Relations.Add(rel)
```

تعبئة السجلات:

كائن البيانات DataSet أصبح جاهزا لملئه بالسجلات، الشيفرة التالية تضيف سجل للجدول الاول والثاني:



```
Dim dr As DataRow = Employees.NewRow
Dim dr2 As DataRow = Sales.NewRow
```

```
dr("المعرف") = 1
dr("الاسم") = "عباس السريع"
```

```
dr2("رقم الموظف") = 1
dr2("دفع نقدي") = True
dr2("المبلغ") = 100
```

```
Employees.Rows.Add(dr)
Sales.Rows.Add(dr2)
```

وهذه بقية السجلات بطريقة مختصرة ومفضلة:



```
Employees.Rows.Add(New Object() {2, "برعي ابو جبهة"})
Employees.Rows.Add(New Object() {3, "عبود اللوح"})
```

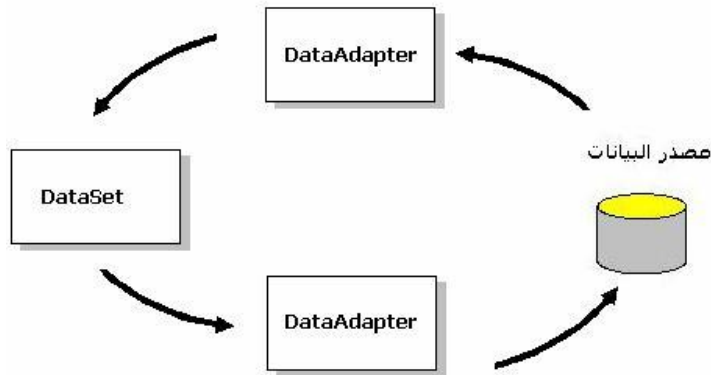
```
Sales.Rows.Add(New Object() {1, True, 200})
Sales.Rows.Add(New Object() {3, False, 150})
Sales.Rows.Add(New Object() {2, True, 120})
```

كائن المحول DataAdapter

في القسم السابق من هذا الفصل، تعرضنا بشكل مختصر إلى الفئة DataSet، يمكنك مراجعة مكتبة MSDN لمعرفة باقي الفئات التي لم اتطرق لها، وفي رأيي الشخصي، لن تستخدم هذه الفئات بكثرة في برامجك الجديدة، إذ أن أغلب استخدامك سيكون مختصر على التعامل مع السجلات فقط، وذلك لأن كائن البيانات DataSet ستحملة من ملف قاعدة البيانات نفسه وسيتأنيك زي البيضة المنشورة!

سيناريو الوضع المنفصل

عند التعامل مع ADO.NET في الوضع المنفصل Disconnected Mode، فالسيناريو يوضحه لك (شكل 18-3) والذي يكون كالتالي: عندما تفتح اتصالاً مع مصدر، اسند مرجع كائن الاتصال Connection إلى كائن DataAdapter، وهو يمثل همزة الوصل بين الكائن DataSet ومصدر البيانات، بعد ذلك ارسل البيانات المراد تعديلها إلى كائن DataSet وبذلك يمكنك إنهاء علاقتك مع مصدر البيانات وإغلاق الاتصال المنشئ بالكائن Connection.



شكل 18-3: سيناريو استخدام كائن المحول DataAdapter مع مصدر البيانات وكائن DataSet.

يمكنك الآن معالجة البيانات وتحريرها وتعديلها عن طريق الكائن DataSet، وتذكر انها لا ترتبط بمصدر البيانات، فجميع العمليات ستتم في جهاز العميل Client، مما لا يشكل اي عمل او مهمة على جهاز مصدر البيانات الخادم Server.

بعد معالجة البيانات -الموجودة في الكائن DataSet- يمكنك ارسالها مرة اخرى إلى كائن DataAdapter والذي بدوره يحدد مصدر البيانات بالعمليات التي قمت بها، تستطيع ارسال الكائن DataSet بعد استقباله بدقة، ساعة، سنة او اي وقت كما يحلو لك. كما لا يشترط الحصول على الكائن DataSet من مصدر بيانات، اذ يمكنك بناء الكائن DataSet من الصفر - كما فعل في القسم السابق من هذا الفصل - وارساله إلى كائن DataAdapter بالرغم من ندرة حدوث هذا الشئ مع التطبيقات الحقيقية.

ميزة عظيمة عليك تقديريها والرفع من شأنها عندما تتعامل في الوضع المنفصل، وهي لا يشترط توافق نوع مزود مصدر البيانات الذي اخذت منه بيانات الكائن DataSet بنفس نوع مزود مصدر البيانات الذي سيتم ارسال بيانات الكائن DataSet بعد معالجتها، لتتمكن -مثلا- من الحصول على السجلات من قاعدة بيانات Microsoft Access®، ثم معالجتها وتحديثها إلى خادم Microsoft SQL Server®.

إنشاء كائن محول DataAdapter

لنبتعد قليلا عن الكلام النظري وننزل إلى مستوى الشيفرات المصدرية، ان كنت تتوي انشاء كائن محول DataAdapter فعليك تحديد نوع المزود المستخدم، فلو كنت تتعامل مع مصدر بيانات ذو مزود من النوع .NET Data Provider OLE DB، فعليك تعريف كائن من الفئة OleDbDataAdapter:

```
Dim msAccessDa As New OleDbDataAdapter()
```

اما ان كنت من مستخدمي المزود .NET Data Provider SQL Server، فالفئة SqlDataAdapter هي التي يفترض دائما استخدامها:

```
Dim msSQLDa As New SqlDataAdapter()
```

المزيد ايضا، كلا الفئتين OleDbDataAdapter و SqlDataAdapter مشتقتان وراثيا من الفئة القاعدية DbDataAdapter (والمشمولة في مجال الاسماء System.Data.Common)، لذلك قد تفيدك هذه الفئة القاعدة من تعريف إجراءات تستقبل وسيطات من مختلف انواع كائنات المحولات DataAdapter:

```
Sub UpdateDate(ByVal da As Common.DbDataAdapter)
...
...
End Sub
```

ملاحظة

الفئة DbDataAdapter هي فئة مجردة Abstract Class، فقد عُرِفَت بالكلمة المحجوزة MustInherit، لذلك لن تتمكن من إنشاء كائنات منها باستخدام New او اي طريقة اخرى بشكل عام:

احلق شواربي ان نجحت!'
Dim x As New DbDataAdapter()

الربط مع اتصال

الخطوة الاولى التي عليك إنجازها (سواء كنت تنوى قراءة او تحديث البيانات)، هي ربط كائن المحول DataAdapter مع كائن اتصال، توجد مجموعة من الاساليب التي تمكنك من عمل ذلك واسهلها ارسال مرجع كائن الاتصال Connection مع مشيد الفئة DataAdapter مع ضرورة ارسال جملة الاستعلام كوسيلة اولى:

```
Dim cn As New OleDbConnection(connString)
cn.Open()
...
Dim da As New OleDbDataAdapter("SELECT * FROM [جدول]", cn)
```

ولا تنسى ضرورة توافق نوع كائن المحول مع نوع كائن الاتصال، فلو كان كائن الاتصال يتبع مزود من النوع SQL .NET Data Provider فقد تم انشائه من الفئة SqlConnection، لذلك عليك الاعتماد على الفئة SqlDataAdapter عوضا عن OleDbDataAdapter للربط الصحيح مع الاتصال:

```
Dim cn As New SqlConnection(connString)
cn.Open()
...
Dim da As New SqlDataAdapter("SELECT * FROM [جدول]", cn)
```

من المهم التنويه هنا، بأن كائن المحول DataAdapter مرتبط ارتباطا وثيقا بكائن الاتصال Connection، فلو تم اغلاق كائن الاتصال بالطريقة Close()، فلن تتمكن من الاستفادة من هذا الكائن (حاله كحال كائن الاوامر Command الذي تحدثت عنه في الفصل السابق).

قد تستغرب الجملة السابقة وتظن انها لا تناسب فكرة ADO.NET للوضع المنفصل، حيث اني ذكرت في بداية هذا القسم من الفصل انك ستعلق الاتصال بمجرد الحصول على البيانات، وانا هنا لا انوي ان أعارض احادتي بل سأزيد توضيحها بقول: يمكنك قطع الاتصال واغلاقه بالطريقة Close() متى ما شئت ولكن بعد تحميل البيانات إلى كائن DataSet.

المزيد ايضا، يمكنك الربط مع اتصال مباشرة دون الحاجة لتعريف كائن Connection بشكل واضح Explicitly، وذلك بارسال نص الاتصال Connection String مع مشيد الفئة:

```
Dim da As New OleDbDataAdapter("SELECT * FROM ... ..", _
"Provider=Microsoft.Jet.OLEDB.4.0;Data Source= ... ..")
```

لا تعتقد ان السطر السابق لا ينشئ كائن اتصال، بل قام بإنشاء كائن اتصال من النوع Connection ايضا ولكنه محضون بداخل الكائن DataAdapter سيتم اغلاقه مباشرة بمجرد اغلاق الكائن الحاضن DataAdapter نفسه.

قراءة البيانات

في هذه الفقرة سأريك كيف تقرأ البيانات، اما في الفقرة القادمة سنقوم بتحديث البيانات إلى مصدر البيانات. وحتى تتمكن من قراءة البيانات، عليك اولا وضع البيانات التي حصلت عليها من كائن المحول DataAdapter إلى كائن البيانات DataSet، يتم ذلك بارسال كائن البيانات DataSet المطلوب إلى الوسيطة الاولى للطريق Fill() والخاصة بالمحول DataAdapter:

```
Dim cn As New OleDbConnection(connString)
cn.Open()
Dim da As New OleDbDataAdapter("SELECT * FROM [جدول الخبايب]", cn)
Dim ds As New DataSet()
...
...
da.Fill(ds, "جدول الخبايب")
```

وبالمناسبة، لست بحاجة الآن لا لكائن الاتصال ولا للاتصال الذي يحمله، فيمكنك الآن إغلاق الاتصال بدون خوف على البيانات:

```
cn.Close()
```

ملاحظة

الوسيلة الثانية للطريقة Fill() السابقة، تمثل اسم الجدول الذي تود ظهوره في الخاصية TableName التابعة للكائن DataTable والمحضون في الكائن DataSet (شكل 18-1).

بمجرد نقل البيانات من كائن المحول DataAdapter إلى كائن البيانات DataSet، فإن البيانات موجودة في جهاز العمل الآن وملكك تستطيع التصرف فيها كما تشاء، وكل ما هو مطلوب منك إيقان الهرم الكائني لكائن البيانات DataSet - رغم إيماني الشديد بأن استخدام الفئة DataRow سيكون كافياً إلى حد كبير لأغلب الأغراض:

```
Dim x As DataRow
For Each x In ds.Tables("جدول الخبايب").Rows
    ListBox1.Items.Add(x("الاسم"))
Next
```

الأخطاء في عملية التحميل:

كما ذكرت مراراً، عند التعامل مع قواعد البيانات، يفضل دائماً تفادي الاستثناءات باستخدام التركيب Try...End Try أو العبارة الأخرى On Error:

```
Try
    da.Fill(ds, "جدول الخبايب")
...
...
Catch ex As Exception
```

```
...
...
...
End Try
```

مع ذلك، تتصحك مستندات .NET Documentation بالاعتماد على الحدث FillError عوضاً عن تقادي الاستثناء بالطرق التقليدية، والسبب في ذلك ان اي عملية خطأ في تعبئة كائن البيانات DataSet (كوجود نوع في مصدر البيانات غير معرف في إطار عمل .NET Framework مثلًا) ستؤدي إلى الغاء عملية تحميل كامل البيانات إلى الكائن DataSet حتى لو كان سبب حدوث الاستثناء حقل واحد.

الحدث FillError التابع للكائن DataAdapter يتم تنفيذه بمجرد حدوث اي خطأ في عملية تعبئة البيانات إلى الكائن DataSet (اي عند استدعاء الطريقة (Fill))، يرسل هذا الحدث وسيطة من النوع FillEventArgs تحتوي على الخاصية Errors التي يمكنك من معرفة الاستثناء ، كما يمكنك اسناد القيمة True إلى الخاصية Continue لإكمال عملية تحميل البيانات إلى كائن البيانات دون توقف:

```
Sub LoadDataSet()
    ...
    ...
    Dim da As New OleDbDataAdapter("... ..", cn)
    Dim ds As New DataSet()

    AddHandler da.FillError, AddressOf FillError
    da.Fill(ds, "جدول الجايب")
    ...
    ...
End Sub

Sub FillError(ByVal sender As Object, ByVal e As FillEventArgs)
    If TypeOf e.Errors Is Exception Then
        ...
        ...
        e.Continue = True
    End If
End Sub
```

تحديث البيانات

يمكنك تحديث مصدر البيانات بكائن بيانات DataSet بعد معالجته بارساله مع الطريقة Update() التابعة للكائن DataAdapter، والتي تتطلب ايضا اسم الجدول في مصدر البيانات:

```
Dim da As New OleDbDataAdapter("[جدول]", cn)
Dim ds As New DataSet()
...
...
da.Update(ds, "جدول")
```

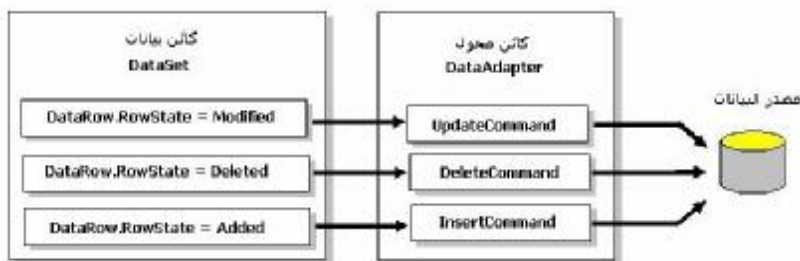
مهلا مهلا يا عزيزي، فالموضوع ليس بالبساطة التي كنت تتوقعها، اذا ان عملية ارسال كائن بيانات DataSet إلى محول DataAdapter قد تتأثر بالتعارضات التي تحدث على السجلات، فمثلا تخيل انك تود تحديث سجل تم حذفه من قبل مستخدم اخر، او انك تنوي اضافة سجل جديد يتعارض رقم مفتاحه الابتدائي Primary Key مع سجل اخر، او انك قد تضيف سجل يختلف في تركيبة حقوله عن تركيبة حقول مصدر البيانات الأساسي.

كما ترى في سطور الفقرة السابقة، نسبة حدوث التعارضات عند عملية تحديث البيانات إلى مصدر البيانات الأساسي كبيرة جدا، وسأنتظر لمعظم حالات هذه التعارضات لاحقا في القسم **اتقاء شر التعارضات**، اما هنا فدعنا نفترض ان هذه التعارضات لم ولن تحدث، ولنتخيل ان مستخدم قاعدة البيانات هو مستخدم واحد فقط، وذلك لتبسيط الامور وتوضيح الاساسيات قبل التعامل مع الحالات المعقدة.

كما اخبرتك قبل قليل، عندما تنوي تحديث مصدر البيانات وارسال السجلات بعد معالجتها ستستخدم الطريقة Update() والتي تعود بعدد السجلات التي تأثرت (تم اضافتها، حذفها، او تعديلها)، ولكن قبل استدعاء هذه الطريقة عليك التعامل مع ثلاث خصائص اخرى تابعة لكائن المحول DataAdapter ايضا هي InsertCommand، UpdateCommand، و DeleteCommand.

حتى تفهم كيف تسند القيم المناسبة للخصائص الثلاث الاخيرة، من الجيد جدا استيعاب طريقة عمل كائن المحول DataAdapter والتي تكون كالتالي: عندما تنوي تحديث البيانات إلى مصدر بيانات باستدعاء الطريقة Update()، سيقوم كائن المحول DataAdapter بالتحقق من الخاصية RowState والتابعة لكل كائن سجل DataRow من سجلات كائن البيانات DataSource، فان كانت Modified سيتم تنفيذ جملة الاستعلام في الخاصية UpdateCommand، وان كانت Deleted سيتم تنفيذ الامر التابع للخاصية

DeleteCommand، اما ان كانت قيمة الخاصية RowState لكائن الحقل هي Added فسيتم تنفيذ جملة الاستعلام SQL في الخاصية InsertCommand والتابعة لكائن المحول DataAdapter (شكل 18-4).



شكل 18-4: تحديد الخاصية المناسبة لتنفيذ جملة الاستعلام.

الخصائص UpdateCommand، InsertCommand، و DeleteCommand ليست خصائص حرفية، وانما خصائص من النوع كائن الأوامر Command (الذي تحدثت عنه في الفصل السابق)، مع ذلك لست بحاجة لاستخدامه، اذ يوفر لك إطار عمل .NET Framework كائن توليد الاوامر SqlCommandBuilder والذي تفصله لك الفقرة الفرعية التالية.

كائن توليد الاوامر SqlCommandBuilder:

الغرض من كائن توليد الامر SqlCommandBuilder هو توليد جمل الاستعلام SQL بحيث توافق العمليات التي تحدثها على السجلات، جمل الاستعلام هذه تتعلق بالاوامر INSERT، UPDATE، و SELECT، وحتى يمكنك استخدام الكائن، أنشئه من الفئة OleDbDataAdapter وارسل مع مشيده كائن المحول DataAdapter:

```
Dim da As New OleDbDataAdapter("...", cn)
Dim cmd As New OleDbCommandBuilder(da)
```

ولا تنسى في حالة استخدامك لكائن محول لمزود من النوع SQL Server .NET Provider، عليك تعريف كائن من الفئة SqlCommandBuilder:

```
Dim da As New SqlDataAdapter("...", cn)
Dim cmd As New SqlCommandBuilder(da)
```

بعد إنشائك لكائن مولد اوامر Commandbuilder، يمكنك استدعاء طرقه GetxxxCommand() لتسندها إلى خصائص كائن المحول DataAdapter الثلاث
:xxxCommand

```
da.UpdateCommand = cmd.GetUpdateCommand
da.DeleteCommand = cmd.GetDeleteCommand
da.InsertCommand = cmd.GetInsertCommand
```

مثال تطبيقي:

دعني اريك مثالا لتحديث سجلات مصدر بيانات في الوضع المنفصل، أنشئ قاعدة باستخدام Microsoft Access®، وأضف جدولاً بالاسم جدول يحتوي على ثلاث حقول هي رقم المعرف، الاسم، والعمر.

سنبدأ بفتح اتصال مع الجدول وقراءة بياناته، ومن ثم استدعاء الطريقة Fill() لكائن المحول DataAdapter حتى نتمكن من تعبئة كائن البيانات DataSet:

```
Dim cn As New OleDbConnection(connString)
cn.Open()
Dim da As New OleDbDataAdapter("SELECT * FROM [جدول]", cn)
Dim ds As New DataSet()

da.Fill(ds, "جدول")
```

الخطوة التالية هي تعديل السجلات، في الشيفرة التالية قمنا بإجراء عملية تعديل للسجل الأول، حذف للسجل الثاني، وإضافة سجل جديد عن طريق كائن البيانات DataSet:

```
With ds.Tables("جدول")
    .Rows(0)("الاسم") = "عباس السريع"
    .Rows(0)("العمر") = 99
End With

ds.Tables("جدول").Rows(1).Delete()
ds.Tables("جدول").Rows.Add(New Object() {Nothing, "برعي ابو جبهة", 55})
```

والان يمكننا تحديث قاعدة البيانات بعد إجراء التعديلات على سجلاتها باستدعاء الطريقة Update()، ولكن عليك اسناد قيم الخصائص xxxCommand قبل استدعائها، يتم ذلك بتعريف كائن مولد الاوامر SqlCommand:

```
Dim cmd As New OleDbCommandBuilder(da)

da.UpdateCommand = cmd.GetUpdateCommand
da.DeleteCommand = cmd.GetDeleteCommand
da.InsertCommand = cmd.GetInsertCommand

da.Update(ds, "جدول")
```

نقطة هامة عليك أخذها بعين الاعتبار، وهي ان كائن مولد الاوامر SqlCommand يعطيك نتائج خاطئة ان كانت اسماء الجداول او الحقول تحتوي على مسافات او حروف خاصة، فالجملة التالية:

```
SELECT * FROM العمليات WHERE رقم المعرف = 4
```

جملة استعلام خاطئة، فهي لا بد ان تكتب الاسماء بين القوسين [و] هكذا:

```
SELECT * FROM [العمليات] WHERE [رقم المعرف] = 4
```

وحتى تضع هذه الاقواس، اسند قيمة في الخاصية QuotePrefix تحدد فيها الحرف الذي سيسبق اسم الجدول والحقل، والخاصية QuoteSuffix الحرف الذي يلي اسم الجدول او الحقل:

```
Dim da As New OleDbDataAdapter("... ..", cn)
Dim cmd As New OleDbCommandBuilder(da)

cmd.QuotePrefix = "["
cmd.QuoteSuffix = "]"

da.UpdateCommand = cmd.GetUpdateCommand
da.DeleteCommand = cmd.GetDeleteCommand
da.InsertCommand = cmd.GetInsertCommand
...
...
...
```

تخصيص أفضل للخصائص xxxCommand

في الفقرة السابقة اعتمدنا على كائن مولد الاوامر SqlCommand لتخصيص اوامر جمل الاستعلام SQL في خصائص كائن المحول xxxCommand، ولكن اكبر عيب من عيوب كائن مولد الاوامر SqlCommand هو ضرورة القيام بمسح على مصدر البيانات للحصول على تركيبة البيانات (اي يقوم كائن الاوامر باستدعاء العبارة SELECT قبل توليد الاوامر) مما يسبب بطئ في عملية التحديث.

ولو كان الامر على المسح الإضافي الذي يجريه لهانت المشكلة، اذ ان خوارزم توليد جمل الاستعلام الذي يتبعه هذا الكائن يشترط وجود حقل لمفتاح ابتدائي Primary Key في الجدول، مما يجعل استخدامه محصور على الجداول التي تحتوي على مفاتيح ابتدائية، ليس هذا فقط، بل يشترك ايضا تضمين حقل المفتاح الابتدائي في نفس جملة الاستعلام SELECT -المستخدمة لحظة إنشاء كائن المحول DataAdapter.

يبقى الحل الاخير هو بناء جمل الاستعلام بنفسك وإسنادها إلى الخصائص xxxCommand والتابعة لكائن المحول DataAdapter، وهو امر يتطلب خبرة منك في تعريف الوسيطات Parameters في جمل الاستعلام SQL.

بالنسبة لجمل الاستعلام والخاصة باضافة الحقول (التي تسندها إلى الخاصية InsertCommand)، فيمكنك الاكتفاء بكائن مولد الاوامر SqlCommand وذلك ان الخوارزم الذي يتبعه يولد بشكل صحيح، اما الحديث عن جمل الاستعلام الاخرى والخاصة بالحذف او التعديل (التي تسندها إلى الخاصيتين UpdateCommand و DeleteCommand)، فعليك انشاء جمل استعلامها بنفسك عن طريق الكائن Command:

```
Dim delSQL As String
Dim updateSQL As String

delSQL = "DELETE FROM [جدول] WHERE [رقم المعرف] = ?"
updateSQL = "UPDATE [جدول] SET [الاسم] = ? , [العمر] = ? WHERE " _
    & "[رقم المعرف] = ?"

Dim cmdDel As New OleDbCommand(delSQL, cn)
Dim cmdUpd As New OleDbCommand(updateSQL, cn)
```

وحتى نخبر الكائن Command أننا استخدمنا وسيطات Parameters في جمل الاستعلام (علامات الاستفهام ? هي الوسيطات)، عليك استخدام المجموعة Parameters Collection لتعريف هذه الوسيطات والحقول التي تمثلها بالاضافة إلى نوع القيم التي تحملها:

```
With cmdDel.Parameters.Add("@p1", GetType(Integer))
    .SourceColumn = "المعرف"
End With

With cmdUpd.Parameters.Add("@p1", GetType(String))
    .SourceColumn = "الاسم"
End With

With cmdUpd.Parameters.Add("@p2", GetType(Integer))
    .SourceColumn = "العمر"
End With

With cmdUpd.Parameters.Add("@p3", GetType(Integer))
    .SourceColumn = "رقم المعرف"
End With
```

والان كل ماهو مطلوب منك اسناد كائنات الاوامر إلى الخصائص المناسبة

```
Dim da As New OleDbDataAdapter("... ..", cn)
...
...
da.DeleteCommand = cmdDel
da.UpdateCommand = cmdUpd
...
...
```

اتقاء شر التعارضات

عملية تحديث البيانات إلى مصادر البيانات ليست مفروشة بالزهور الوردية، بل ان حقائق الشوك احد سماتها، وكل ما فعلناه في الفقرات السابقة كان بافتراض ان كل شيء يتم على ما يرام، فعندما تحدث او تحذف سجل فانك متأكد من ان ذلك السجل موجود، وعندما تضيف سجل جديد فالمفتاح الابتدائي Primary key لا يتعارض مع اخر.

ولكن عندما يتعدد المستخدمين لقاعدة البيانات، فعليك -ثبتت ان أبيت- من اتقاء شر التعارضات، واسهل ما يمكنك القيام به هو استدعاء الطريقة Update() بين فكي التركيب Try ... End Try

```
Try
    ...
    da.Update(ds, "جدول")
Catch
    ...
End Try
```

يعيب الأسلوب السابق، ان اي خطأ في عملية تحديث احد السجلات إلى مصدر البيانات سيوقف عملية التحديث لكافة السجلات الأخرى (والموجودة في كائن البيانات DataSet)، لذلك قد تسند القيمة True إلى الخاصية ContinueUpdateOnError حتى تمكن كائن المحول من متابعة تحديث باقي السجلات ان حدث خطأ في عملية تحديث احدها:

```
da.ContinueUpdateOnError = True
da.Update(ds, "جدول")
```

ويمكنك معرفة ان حدث فعلا خطأ لحظة التحديث عن طريق الخاصية HasChanges والتابعة لكائن البيانات DataSet - وليس المحول DataAdapter:

```
If ds.HasChanges() Then
    ...
    ...
End If
```

يمكنك الاستعانة بالحدث RowUpdated التابع لكائن المحول DataAdapter، والذي يتم تنفيذه بعد عملية تحديث كل سجل من السجلات، الا انك لن تحتاج اليه في اغلب الاحوال، حيث ساعرض لك في الفقرة التالية طريقة يمكنك اتباعها لمعرفة السجلات التي حدثت بها التعارضات.

عرض التعارضات

الاسلوب الاول البرمجي مفضل لدى اغلب المبرمجين، والسبب هو توفير عناء كتابة الشيفرة المصدرية وازالة العبء عنهم، بل وإعطاء المستخدم الحرية المطلقة في عمل ما يريد. الفكرة من هذا الاسلوب ببساطة هي عدم فعل اي شيء وابقاء السجلات الغير محدثه كما هي، وكل ما يتطلبه منك كتابة نص - او اي شيء اخر - في الخاصية RowError والتابعة للكائن DataRow، ومن المهم استدعاء الطريقة RejectChanges() للسجلات التي كنت ترغب في حذفها وفشلت، وذلك حتى لا تختفي من كائن الجدول DataTable التابع لكائن مصدر البيانات DataSet:

```
Dim da As OleDbDataAdapter
Dim ds As DataSet
...
...

da.ContinueUpdateOnError = True
da.Update(ds, "جدول")
```

```

If ds.HasChanges Then
    Dim dr As DataRow

    For Each dr In ds.Tables("جدول").Rows
        If dr.RowState = DataRowState.Added Then
            dr.RowError = "خطأ أثناء اضافة هذا السجل"
        ElseIf dr.RowState = DataRowState.Modified Then
            dr.RowError = "خطأ أثناء تعديل هذا السجل"
        ElseIf dr.RowState = DataRowState.Deleted Then
            dr.RowError = "خطأ أثناء حذف هذا السجل"
            ' استدعي هذه الطريقة حتى لا يتم
            ' حذف السجل من كائن البيانات
            ' DataSet
            dr.RejectChanges()
        End If
    Next
End If

```

الحدث RowUpdated

عملية تصحيح التعارضات تلقائياً عملية ليست مستحيلة، ولكنها تعتمد اعتماد كلي على أسلوبك ومتطلبات المشروع الذي تتجزه، فبعض المبرمجين سيقومون بنقل السجلات المتعارضات إلى جدول خاص بها، وآخرون قد يحفظونها في ملفات نصية، وهناك من سيحاول اضافة سجل جديد ونسخه ان تعارضت عملية تحديث سجل مع مستخدم آخر. لذلك، دعني اعود إلى الحدث RowUpdated واخبرك كيف يمكنك الاستفادة منه لحل مشاكل التعارضات بنفسك:

```

Dim WithEvents da As OleDbDataAdapter

Private Sub da_RowUpdated(ByVal sender As Object, _
    ByVal e As System.Data.OleDb.OleDbRowUpdatedEventArgs) _
    Handles da.RowUpdated
    ...
    ...
End Sub

```

الحدث RowUpdated يتم تنفيذه في كل مرة يتم فيها تحديث سجل وإرساله إلى مصدر البيانات (سواء نجحت عملية التحديث أو فشلت للسجل)، الوسيطة التي يرسلها الحدث تحتوي على مجموعة من الخصائص، منها الخاصية StatementType والتي تعود بقيمة تمثل نوع العملية التي أجريت على السجل فيما لو كانت Insert اضافة للسجل، Delete حذف، أو Update تحديث. وبالنسبة للقيمة Select فلا تستخدمها لأنها لن تحدث.

الخاصيتين Command و Row تعودان بمرجع لكائنين، الاولى من نوع كائن الاوامر Command يمثل الامر الذي استخدم لحظة تحديث السجل، وهو نفس الكائن الموجود في الخصائص UpdateCommand، DeleteCommand، و InsertCommand. اما الخاصية Row فهي تعود بكائن من النوع DataRow يمثل السجل الذي تم تحديثه. بالنسبة للخاصية RecordsAffected فهي تعود بعدد السجلات التي تأثرت لحظة التحديث، ومن البديهي ستكون قيمتها 1 ان تم كل شيء على ما يرام، او صفر ان فشلت عملية تحديث السجل. اما الخاصية Errors فهي ستعود بكائن استثناء Exception يفيدك كثيرا لمعرفة تفاصيل اكثر عن الخطأ الذي منع عملية تحديث السجل، وان كان نوع هذا الاستثناء هو DBConcurrencyException فذلك يعني حدوث تعارض، وغير ذلك فقد يكون خطأ في عدم وجود صلاحيات لك في نفس مصدر البيانات.

اهم خاصية من خصائص وسيطة الحدث RowUpdated هي الخاصية Status، والتي يمكنك اسناد القيمة Continue لها ان اردت اكمال عملية التحديث، القيمة ErrorsOccurred والتي تجعل الحدث يعبر عن فشل عملية تحديث السجل كخطأ، القيمة SkipCurrentRow تتجاهل عملية تحديث السجل، او القيمة SkipAllRemainingRows التي تتجاهل عملية تحديث السجل الحالي وكافة السجلات اللاحقة.

كم رأيت، التعامل مع فئات ADO.NET في الوضع المنفصل ليس بالأمر الهين ويتطلب منك الحذر الشديد رغم انه يعطيك مرونة كبيرة في عملية نقل البيانات من مصدر بيانات إلى اخر، كما انك ستعتمد عليه ايضا ان كنت تود ربط أدوات Windows Forms بمصادر البيانات كما سنرى في الفصل التالي.

ربط البيانات والتكامل مع XML

يمكنك عرض البيانات على المستخدم يدويا باستخدام الفئات السابق ذكرها، إلا ان عملية ربط البيانات بالأدوات ستوفر عليك الكثير من الجهد والوقت. كما تكامل ADO.NET مع لغة وصف البيانات XML سيوفر عليك عشرات الشيفرات ان اردت تبادل البيانات مع التطبيقات الأخرى سواء في الشبكة المحلية أو على الانترنت أو بشكل خدمات ويب Web Services. في هذا الفصل سأنتقل إلى موضوعين مستقلين، الأول يختص بربط أدوات Windows Forms مع مصادر البيانات، والثاني يتحدث عن سمات XML التي يمكن ان تجنيها عند التعامل مع ADO.NET.

ربط البيانات

ربط البيانات **Data Binding** هي عملية ربط مصدر بيانات Data Source بأداة من أدوات Windows Form بحيث تمكن المستخدم من قراءة مصدر البيانات وتحديثه أيضا عن طريق الأدوات (كالأداة TextBox، ListBox... الخ). ربط مصادر البيانات بأدوات Windows Forms سيسهل عليك الكثير ويوفر عليك عناء كتابة الشيفرات المصدرة الطويلة، فكل ما هو مطلوب منك اعداد بضعة خصائص للأدوات وربطها مع مصدر البيانات، وستصبح النافذة موجه إلى مصدر البيانات، وتتمكن المستخدم أيضا من التنقل بين سجلاتها بالطرق التقليدية (كإضافة زر التالي Next، السابق Back، مؤشر رقم السجل الحالي،... الخ) (شكل 19-1).

شكل 19-1: ربط الادوات بمصدر بيانات.

أنواع الربط

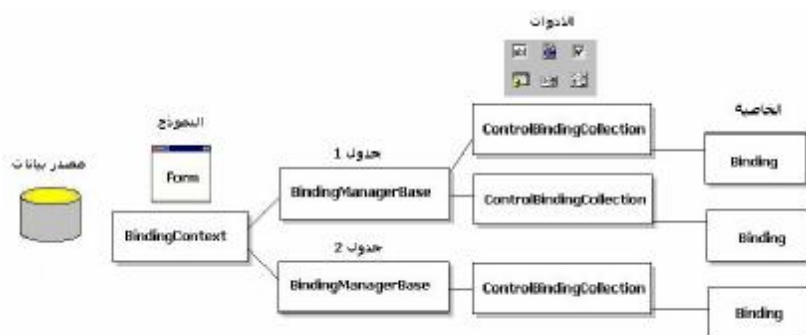
من منظور تحديث البيانات إلى مصادر البيانات، فيوجد نوعين من الربط هما **ربط البيانات أحادي الاتجاه One-way data binding**، و**ربط البيانات ثنائي الاتجاه Two-way data binding**. في الربط الأحادي فإن الأداة سيتم ربطها بمصدر البيانات على شكل للقراءة فقط **ReadOnly**، ولن يتم تحديث مصدر البيانات عند إجراء أي تعديل على محتويات الأداة المربوطة به. وفي الربط ثنائي الاتجاه، فإن الأدوات سيتم ربطها بنسق قابل للقراءة والكتابة، وأي تعديل في محتوى الأداة المربوطة سيتم إبلاغ وتحديث مصدر البيانات بشكل تلقائي.

أما من منظور عدد الخصائص والأدوات المربوطة بمصدر البيانات، فيمكننا أن نصنفها إلى نوعين هما **الربط البسيط Simple Binding** و**الربط المعقد Complex Binding**، في الربط البسيط فإنك تربط خاصية واحدة للأداة بحقل مكافئ لها بمصدر البيانات، كأن تربط سُمِّلا-الخاصية **Text** لأداة **TextBox** بحقل الاسم في مصدر البيانات. أما في الربط المعقد، فإنك تربط مجموعة قيم وخصائص للأداة مع أكثر من حقل (أو أكثر من سجل) في مصدر البيانات، كأن تربط سُمِّلا- المجموعة **Items** والخاصية بالأداة **ListBox** بمجموعة سجلات تمثل المبيعات التي قام بها الموظف.

ميزة أخرى تجنيها من ربط البيانات بالأدوات وهي عدم حصر عملية الربط على نوع معين من مصادر البيانات، بل يمكن لأي كائن يحتوي على الواجهة **IList** أن تقوم بربطه، كان تربط مصفوفة **Array** بالأدوات، أو تستخدم كائنات **ADO.NET** لربط بقواعد البيانات -كما سترى لاحقا. ودعنا الآن نرى كيفية عمل الربط حتى تستوعب الفكرة بالفقرة التالية.

ميكانيكية الربط

يمكنك استخدام بيئة التطوير .NET Visual Studio لربط أدواتك مع مصدر البيانات باستخدام الفأرة، إلا ان استيعاب ميكانيكية الربط أمر في غاية الأهمية حتى تتقن استخدام الشيفرة والكائنات التي تنجزها (شكل 19-2).



شكل 19-2: ميكانيكية ربط البيانات Data Binding.

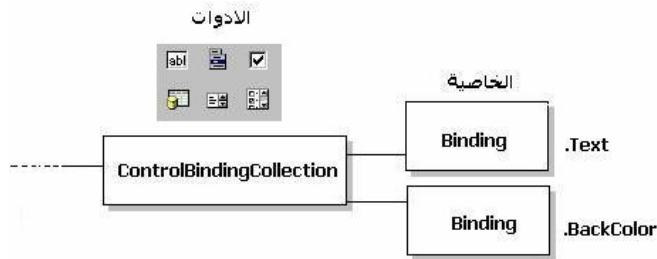
لنبدأ من أقصى اليسار ونحدث عن الكائن BindingContext، الخاصية التي تحمل نفس اسم الكائن والتابعة لنافذة النموذج (أو الاداة الحاضنة ايضا) تعود بكائن من هذا النوع، والذي يمثل رابط لمجموعة من الادوات مع مصدر البيانات، والحاجة اليه تتضح ان علمت ان نافذة النموذج قد يتم ربطها مع اكثر من مصدر بيانات.

الخاصية BindingContext السابقة تتطلب وسيطة تمثل مصدر البيانات، لتعود بكائن من النوع BindingManagerBase، وهو الحاضن أو الرابط الرئيسي للادوات مع مصدر البيانات، يمكن الاستفادة منه -مثلا- في تعيين قيمة لخاصيته Position، والتي تمكنك من تغيير موقع السجل الحالي الذي تعرضه الادوات.

تحتوي كل اداة من ادوات Windows Forms على الخاصية DataBindings، والتي تمثل مجموعة من النوع ControlBindingCollection. في هذه المجموعة، تستطيع اضافة، حذف، أو تعديل كائنات الربط والتي من النوع Binding، مع العمل ان يمكن للداة ان تحتوي على اكثر من كائن ربط.

كائن الربط Binding هو حجر الاساس في عملية ربط خاصية واحدة فقط بحقل واحد فقط بمصدر البيانات، وان اردت ربط اكثر من خاصية باكثر من حقل في نفس الاداة، عليك انشاء كائن ربط جديد.

من الكلام السابق، يتضح لنا انه يمكن للاداة الواحدة من ان تربطها باكثر من حقل وذلك بتكوين علاقة بين احد خصائصها والحقل المطابق لها في مصدر البيانات، وذلك في عدد كائنات الربط Binding التابعة للمجموعة ControlBindingCollection والخاصة بكل اداة، فيمكنك مثلا انشاء كائنين ربط بين الخاصية Text والحقل CarName، والخاصية Backcolor والحقل CarColor بنفس الاداة ونفس مصدر البيانات (شكل 19-3).



شكل 19-3: ربط أكثر من خاصية للأداة بأكثر من حقل.

الربط إلى مصفوفة

في اغلب برامجك الجديدة، فانك ستقوم بربط الادوات مع مصادر بيانات لقواعد بيانات تقليدية، ويمكنك عمل ذلك باستخدام فئات ADO.NET، ولكنني فضلت إعطائك طريقة ربط الادوات بمصادر البيانات من النوع قيم لمصفوفات حتى تسهل عليك عملية استيعابها، عرف الفئة التالية:

```

Class Person
    Private m_name As String
    Property Name() As String
        Get
            Return m_name
        End Get
        Set(ByVal Value As String)
            m_name = Value
        End Set
    End Property

```

```

Private m_age As Integer
Property Age() As Integer
    Get
        Return m_age
    End Get
    Set(ByVal Value As Integer)
        m_age = Value
    End Set
End Property

Private m_married As Boolean
Property Married() As Boolean
    Get
        Return m_married
    End Get
    Set(ByVal Value As Boolean)
        m_married = Value
    End Set
End Property

Sub New(ByVal name As String, ByVal age As Integer, _
    ByVal married As Boolean)

    Me.Name = name
    Me.Age = age
    Me.Married = married
End Sub
End Class

```

يمكنك تعبئة مصفوفة من الفئة السابقة بقرائها من ملف نصي أو كتابتها برمجيا:



```

Public Class Form1
    Inherits System.Windows.Forms.Form
    ...
    ...
    Dim Persons() As Person = {New Person("A", 1, True), _
        New Person("B", 2, False), New Person("C", 3, True)}
End Class

```

اضف أداتي TextBox واداة CheckBox في نافذة النموذج، واصل كائن ربط Binding في كل خاصية DataBindings لكل اداة، يمكنك تعريف كائن ربط باستخدام الكلمة المحجوزة New، أو اضافته مباشرة بالطريقة Add() التابعة للخاصية DataBindings (وهي المجموعة ControlBindingCollection)، وفي كلا الحالتين فيطلب كائن الربط Binding اسم الخاصية ومصدر البيانات واسم الحق في مصدر البيانات:



```
Public Class Form1
    Inherits System.Windows.Forms.Form
    ...
    ...
    Private Sub Form1_Load(ByVal sender As System.Object, _
        ByVal e As System.EventArgs) Handles MyBase.Load

        Dim b As New Binding("Text", Persons, "Name")

        txtName.DataBindings.Add(b)
        txtAge.DataBindings.Add("Text", Persons, "Age")
        CheckBox1.DataBindings.Add("Checked", Persons, "Married")
    End Sub
End Class
```

بهذا تكون انتهيت من عملية ربط الادوات بمصدر البيانات، مع ذلك قد تحتاج إلى اضافة ازرار Button لتمكن المستخدم من التنقل بين السجلات، يتم ذلك باسناد القيمة المناسبة للخاصية Position والتابعة للكائن المحضون BindingManagerBase والذي تصل اليه من الخاصية :BindingContext



```
Public Class Form1
    Inherits System.Windows.Forms.Form
    ...
    ...
    Private Sub cmdNext_Click(ByVal sender As System.Object, _
        ByVal e As System.EventArgs) Handles cmdNext.Click

        Me.BindingContext(Persons).Position += 1
    End Sub

    Private Sub cmdBack_Click(ByVal sender As System.Object, _
        ByVal e As System.EventArgs) Handles cmdBack.Click

        Me.BindingContext(Persons).Position -= 1
    End Sub

    Private Sub cmdLast_Click(ByVal sender As System.Object, _
        ByVal e As System.EventArgs) Handles cmdLast.Click

        Me.BindingContext(Persons).Position = _
            Me.BindingContext(Persons).Count - 1
    End Sub
```

```
Private Sub cmdFirst_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles cmdFirst.Click

    Me.BindingContext(Persons).Position = 0
End Sub
End Class
```

الربط باستخدام ADO.NET

عملية الربط باستخدام ADO.NET لا تختلف كثيرا عن ما فعلنا بالمصفوفة Persons السابقة، فكل ما هو مطلوب منك استبدال مصدر البيانات ووضع الكائن DataSet:

```
Dim cn As New OleDbConnection(connString)
cn.Open()
Dim da As New OleDbDataAdapter("SELECT * FROM [جدول]", cn)
Dim ds As New DataSet()
da.Fill(ds, "جدول")
cn.Close()

txtName.DataBindings.Add("Text", ds, "الاسم.الاسم")
```

صحيح ان عملية الربط هي ثنائية الاتجاه Two-way data binding، ولكن عملية التحديث ستم على كائن البيانات DataSet فقط، وذلك هو اسلوب ADO.NET للوضع المنقطع Disconnected Mode، ويبقى الباقي للقيام بعملية التحديث الفعلية في مصدر البيانات باستدعاء الطريقة Update() لكائن المحول DataAdapter - كما تعلمنا سابقا في الفصل الثامن عشر ADO.NET للوضع المنقطع.

ملاحظة

تمكنك بيئة التطوير Visual Studio .NET من توليد الشيفرات الضرورية لربط الادوات بمصادر البيانات دون الحاجة لكتابتها بنفسك، وذلك بفتح نافذة مستكشف الخوادم Server Explorer (تصل اليها من قائمة View)، ومن ثم تحديد الجدول المراد ربط وسحبه وإلقائه Drag & Drop إلى نافذة النموذج، لتستخدم الفأرة فقط في ربط الادوات مع مصدر البيانات (راجع مستندات .NET Documentation. لمزيد من التفاصيل).

المزيد ايضا، يحتوي الكائن BindingManagerBase (شكل 19-2)، على مجموعة كبيرة من الطرق والخصائص، كالطريقة AddNew() لإضافة سجل جديد، الطريقة CancelCurrentEdit() لإلغاء التحديثات، والطريقة RemoveAt() لحذف سجل، كما يمكنك فنص حدثه PositionChanged الذي يتم تنفيذه بمجرد انتقال المؤشر إلى سجل آخر:

```
Dim WithEvents Bmb As BindingManagerBase

Private Sub Form1_Load(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles MyBase.Load
    ...
    ...
    Dim ds As New DataSet()
    ...
    ...
    Bmb = Me.BindingContext(ds)
End Sub

Private Sub Bmb_PositionChanged(ByVal sender As Object, _
    ByVal e As System.EventArgs) Handles Bmb.PositionChanged

    ' اكتب الكود هنا '
    ...
    ...
End Sub
```

الربط المعقد Complex Binding

ان عمليات الربط التي قمنا بها في الفقرات السابقة، تصنف من ضمن الربط البسيط Simple Binding، وذلك لان ميكانيكية الربط تشمل خاصية واحدة لحقل واحد -حتى وان تعددت الخصائص المربوطة بأكثر من حقل في نفس الأداة.

اما الربط المعقد Complex Binding فلا تدعمه كافة الأدوات، وان أردت تطبيقه فعليك استخدام أداة من الأدوات ListBox، ComboBox، أو DataGrid، بحيث تمكن من ربط الأداة بأكثر من سجل أو أكثر من حقل.

أكثر ما يفيدك الربط المعقد لمصادر البيانات لحظة العلاقات Relationships بين الجداول، فافترض مثلا ان لدينا جدول الموظفين يحتوي على مجموعة من حقول كاسم الموظف، الراتب، العمر، وغيرها، بالإضافة إلى حقل يمثل القسم الذي يعمل فيه الموظف وهذا الحقل سيكون مربوط بعلاقة مع جدول آخر يمثل الاقسام وأسمائها.

إن أردت ربط الأدوات بالجدول الاول، فقد تتبع أسلوب الربط البسيط Simple Binding، ولكن يعيبه ان المستخدم سيضطر إلى كتابة رقم القسم الذي يعمل فيه الموظف والمشمول في

العلاقة مع جدول الاقسام (شكل 4-19 A). اما ان اردت التسهيل عليه، فيمكنك استخدام الاداة ComboBox التي تعرض جميع سجلات جدول الاقسام، وتمكن المستخدم ايضا من تحديد القسم الذي يعمل فيه الموظف باستخدام الفأرة ودون الحاجة إلى كتابة رقم القسم يدويا (شكل 4-19 B).



شكل 4-19: استخدام الاداة ComboBox للربط المعقد.

ان اردت استخدام الاداة ComboBox بهذه الطريقة، فعليك اسناد القيم المناسبة لثلاث خصائص هي: DisplayMember، ValueMember، و DataSource. في الخاصية الاولى تحدد الحقل الذي تود عرض سجلاته في الاداة (سيكون اسم القسم في مثالنا)، وفي الثانية تحدد فيها الحقل الذي تود استخدام قيمته الفعلية (سيكون رقم القسم)، اما الاخير فتحدد فيها كائن مصدر البيانات. قد تسند الخصائص السابق بهذا الشكل:

```
Dim da2 As New OleDbDataAdapter("SELECT * FROM [جدول الاقسام]", cn)
Dim ds2 As New DataSet()
da2.Fill(ds2, "جدول الاقسام")
```

```
ComboBox1.DisplayMember = "[جدول الاقسام].[اسم القسم]"
ComboBox1.ValueMember = "[جدول الاقسام].[رقم القسم]"
ComboBox1.DataSource = ds2
```

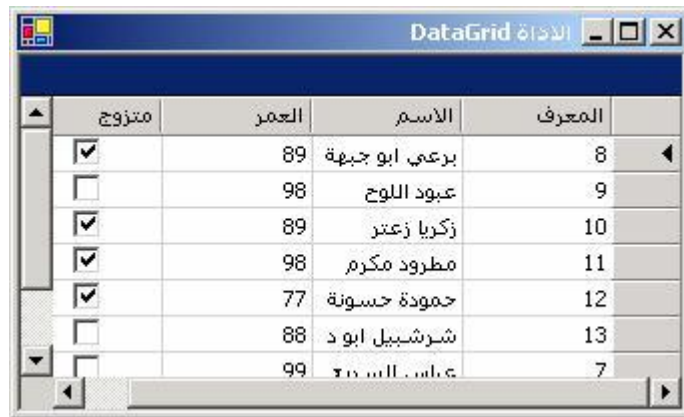
بعد اسنادك للقيم المناسبة، تأتي عملية ربط اخرى لنفس الاداة ComboBox1 بمصدر البيانات الاول (جدول الموظفين) وذلك لاننا نريد تغيير قيمة حقل "رقم القسم" في الجدول الاول بالاختيار الذي يحدده المستخدم في الخاصية SelectedValue:

```
' ds هو كائن مصدر البيانات الاول
ComboBox1.DataBindings.Add("SelectedValue", ds, _
    "[جدول الموظفين].[رقم القسم]")
```

اجمل ربط معقد تتجزه لك الاداة DataGrid بحيث تعرض لك كامل جدول كائن البيانات DataSet بكافة حقوله وسجلاته، وكل ما تطلبه منك هو مصدر البيانات في الخاصية DataSource، والجدول المراد ربطه في الخاصية DataMember (شكل 5-19):

```
Dim da As New OleDbDataAdapter("SELECT * FROM [جدول]", cn)
Dim ds As New DataSet()
...
...
da.Fill(ds, "جدول")

DataGrid1.DataSource = ds
DataGrid1.DataMember = "جدول"
```



شكل 5-19: الربط المعقد باستخدام الأداة DataGrid.

فئات خاصة بـ XML

في الحقيقة، لست بحاجة إلى استخدام فئات خاصة بلغة وصف البيانات XML ان اردت الاستفادة من سمات XML والتي توفرها لك فئات ADO.NET، إلا أنني فضلت عرض بضعة فئات من اطار عمل .NET Framework. قد تحتاج لها يوما من الايام عند تعاملك مع XML.

ملاحظة

عند استخدام فئات XML قد تحتاج إلى استيراد مجال الاسماء التالي:

Imports System.XML

كما يمكنك قراءة الملحق أ: **لغة وصف البيانات XML** في نهاية هذا الكتاب، للحصول على مقرر سريع حول لغة وصف البيانات XML إن كنت تجهلها.

الفئة XmlTextReader

كما كنا سابقا -في الفصل الثامن **الملفات والمجلدات** - نعتمد على الفئة StreamReader لقراءة الملفات النصية، والفئة BinaryReader للملفات الثنائية، فإننا سنعتمد على الفئة XmlTextReader ان اردت فتح وقراءة الملفات بصيغة XML.

مع ذلك، طريقة التعامل مع كائن XmlTextReader ليس كالتعامل مع كائنات StreamReader أو BinaryReader، وانما يشبه إلى حد كبير التعامل مع الكائن DataReader والذي استخدمناها للوضع المتصل Connected Mode في الفصل السابع عشر **استخدام ADO.NET**.

ان كان الكائن DataReader يتطلب كائن Command ليتم إنشائه، فان الكائن XmlTextReader لا يحتاج منك سوى اسم الملف ترسله مع مشيده:

```
Dim xmlData As New XmlTextReader("C:\mydata.xml")
```

وكما تعلم، ملفات XML ما هي إلا ملفات بيانات أشبه بجداول قواعد البيانات، ستستعلم عنها في حلقة Do ... Loop لتستدعي الطريقة Read() -تماما كما فعلت مع الكائن DataReader:

```
Dim xmlData As New XmlTextReader("C:\mydata.xml")
Do While xmlData.Read
    ...
    ...
Loop
```

يفضل دائما معرفة نوع القيمة التي تقرأها من ملف XML، كأن تكون عنصر جذري للملف XML Document، أو عنصر XML Element، قيمة سجل، واصفة XML Attribute، تعليق Comment... الخ، عن طريق الخاصية NodeType:

```
Do While xmlData.Read
    Select Case xmlData.NodeType
        Case XmlNodeType.Document
            ...
            ...
        Case XmlNodeType.Attribute
            ...
            ...
        Case XmlNodeType.Element
            ...
            ...
        Case XmlNodeType.Entity
            ...
            ...
    End Select
Loop
```

تستطيع قراءة العنصر أو أي قيمة تقرأها عن طريق الخاصية Name:

```
Do While xmlData.Read
    If xmlData.NodeType = XmlNodeType.Element Then
        MsgBox ("<" & xmlData.Name & ">")
    End If
Loop
```

اخيرا، لا تنسى إغلاق الملف دائما باستدعاء الطريقة Close():

```
xmlData.Close()
```

الفئة XmlTextWriter

تتشئ كائنات من الفئة XmlTextWriter ان اردت كتابة ملفات XML، واستخدامها سهل جدا ولا يحتاج إلى تفصيل، فكل ما هو مطلوب منك ارسال اسم الملف وصفحة المحارف Encoding مع المشيد:

```
Dim xmlData As New XmlTextWriter("C:\Test.xml", _
    System.Text.Encoding.UTF8)
```

ومن ثم البدء باستدعاء الطريقة WriteStartDocument حتى تتمكن من البدء في تعبئة الملف:

```
xmlData.WriteStartDocument()
```

الآن تستطيع كتابة كل ما تريده في الملف باستخدام عشرات الطرق المتشابهة والتي تكون صيغتها WriteStartxxx() و WriteEndxxx()، الطريقة الاولى تبدأ الكتابة في المستوى الفرعي، والآخرى تقوم باغلاق المستوى حتى تتم الكتابة إلى نفس المستوى الحالي، المثال التالي يكتب مجموعة من العناصر XML Elements:

```
xmlData.WriteStartElement("Table")

xmlData.WriteStartElement("Name")
xmlData.WriteString("تركي العسيري")
xmlData.WriteEndElement()

xmlData.WriteStartElement("Name")
xmlData.WriteString("عبد الله ابراهيم")
xmlData.WriteEndElement()
xmlData.WriteEndElement()

xmlData.Close()
```

مخرجات الشيفرة السابقة ستكون:

```
<?xml version="1.0" encoding="utf-8" ?>
<Table>
  <Name>تركي العسيري</Name>
  <Name>عبد الله ابراهيم</Name>
</Table>
```

راجع مكتبة MSDN للحصول على تفاصيل كافة الطرق الأخرى.

تكامل XML و ADO.NET

في هذا القسم سأريك كيف يمكنك الاستفادة من دعم ADO.NET للغة وصف البيانات XML، وعرض كيف نقرأ من مصادر البيانات لنتخرجها على نسق XML، وفي المقابل أيضا كيف ترسل البيانات بنسق XML إلى مصادر البيانات.

كتابة البيانات بصيغة XML

كل ما هو مطلوب منك لكتابة البيانات من مصادر البيانات بصيغة XML، استدعاء الطريقة WriteXml() التابعة لكائن البيانات DataSet:

```
Dim cn As New OleDbConnection(connString)
cn.Open()

Dim daEmp As New OleDbDataAdapter("SELECT * FROM [Employees]")
Dim daDep As New OleDbDataAdapter("SELECT * FROM [Departments]")
Dim ds As New DataSet()

daEmp.Fill(ds, "Employees")
daDep.Fill(ds, "Departments")
cn.Close()

ds.Relations.Add("علاقة", ds.Tables("Departments").Columns("ID"), _
    ds.Tables("Employees").Columns("DepartmentID"))

ds.WriteXml("C:\test.xml")
```

الطريقة السابقة ستحول جميع الجداول، السجلات، والحقول التابعة لكائن البيانات DataSet إلى ملف بصيغة xml والذي قد تكون مخرجاته شيئاً مثل:

```
<?xml version="1.0" encoding="utf-8" ?>
<NewDataSet>
  <Employees>
    <Name>خالد ابراهيم</Name>
    <Age>99</Age>
    <DepartmentID>1</DepartmentID>
  </Employees>
  <Employees>
    <Name>عمر عبدالله</Name>
    <Age>88</Age>
    <DepartmentID>2</DepartmentID>
  </Employees>
  <Employees>
    <Name>احمد محمد</Name>
    <Age>77</Age>
    <DepartmentID>2</DepartmentID>
  </Employees>
  ...
  <Departments>
    <ID>1</ID>
    <Name>قسم المبيعات</Name>
  </Departments>
  <Departments>
    <ID>2</ID>
```

```
<Name>قسم التسويق</Name>
</Departments>
...
...
</NewDataSet>
```

ان امعنت النظر في شيفرة XML السابقة، سيتضح لك انه يوجد جدولين Employees و Departments، وتم عرض مخرجات سجلاتهم بشكل مستقل رقم وجود علاقة تربط بينهما، لذلك قد تفضل عرض السجلات بشكل متداخل Nested حتى تتداخل السجلات التابعة مع المتبوعة، لعمل ذلك عد إلى كائن البيانات DataSet واسند القيمة True للخاصية Nested والتابعة لكائن العلاقة:

```
ds.Relations("علاقة").Nested = True
ds.WriteXml("C:\test.xml")
```

العناصر ستكون متداخلة بهذا الشكل:

```
<?xml version="1.0" encoding="utf-8" ?>
<NewDataSet>
...
<Departments>
  <ID>1</ID>
  <Name>قسم المبيعات</Name>
  <Employees>
    <Name>خالد ابراهيم</Name>
    <Age>99</Age>
    <DepartmentID>1</DepartmentID>
  </Employees>
  ...
  ...
</Departments>

<Departments>
  <ID>2</ID>
  <Name>قسم التسويق</Name>
  <Employees>
    <Name>عمر عبدالله</Name>
    <Age>88</Age>
    <DepartmentID>2</DepartmentID>
  </Employees>
  <Employees>
    <Name>احمد محمد</Name>
    <Age>77</Age>
    <DepartmentID>2</DepartmentID>
  </Employees>
  ...
  ...
</Departments>
```

```

</Departments>
...
...
...
</NewDataSet>

```

قراءة البيانات بصيغة XML

وكما هو الحال مع كتابة البيانات بالطريقة WriteXml() يمكنك قراءة البيانات من أي ملف XML وإرساله إلى كائن بيانات DataSet باستدعاء الطريقة ReadXml فقط:

```

Dim ds As New DataSet
ds.ReadXml ("C:\test.xml")

```

كائن البيانات DataSet السابق يحمل كل السجلات الموجودة في الملف test.xml، قد تحتاج إلى إعادة تسمية الحقول والجداول ان لم تتوافق مع مصدر البيانات الذي تنوي إرساله اليه عن طريق كائن المحول DataAdapter.

بهذا نكون قد وصلنا إلى نهاية الجزء الرابع **برمجة قواعد البيانات**، وتعلمنا أساسيات استخدام فئات ADO.NET للوضعين المتصل والمنفصل، يتبقى الأمر عليك ان اردت استخدام فئات ADO.NET بالشكل الامثل ومعرفة كافة الطرق والخصائص التي لم اتطرق لها من مكتبة MSDN. حالياً، يمكنك العودة إلى نماذج Windows Forms لاستخدام ADO.NET معها أو المضي قدماً إلى **برمجة ويب** عنوان الجزء الخامس والأخير من هذا الكتاب.

الجزء الخامس

برمجة ويب

تطبيقات ASP.NET (1)

ان كنت احد اعضاء موقع شبكة المطورون العرب، ستلاحظ ان اسم الدخول الخاص بك سيظهر في يمين صفحات الموقع، وان كنت تعتقد اننا قمنا بتصميم صفحة خاصة لكل عضو، فاعتقادك ليس في محله، اذ كل ما فعلناه هو تصميم صفحة واحدة تختلف باختلاف اعدادات زائر الموقع. في هذا الفصل ستكون بدايتك لتطوير تطبيقات ASP.NET، وسيتمحور هذا الفصل حول نماذج Web Forms وطريقة عملها واطراف الادوات عليها وكتابة الشيفرة بها، وسيكون مدخلك الابتدائي لتطوير تطبيقات ASP.NET، اما الفصل القادم فسنحدث عن تطوير تطبيقات ASP.NET بشكل عام.

ملاحظة

يتطلب هذا الفصل معرفة مسبقة بأغلب مصطلحات الانترنت وتصميم المواقع الشائعة، كما يفترض إلمامك التام بلغة تنسيق البيانات HTML، ويفضل ان تكون لديك خبرة سابقة في برمجة ويب (وكأني منزل إعلان وظيفة شاعرة في احد الصحف!).

الخادم IIS

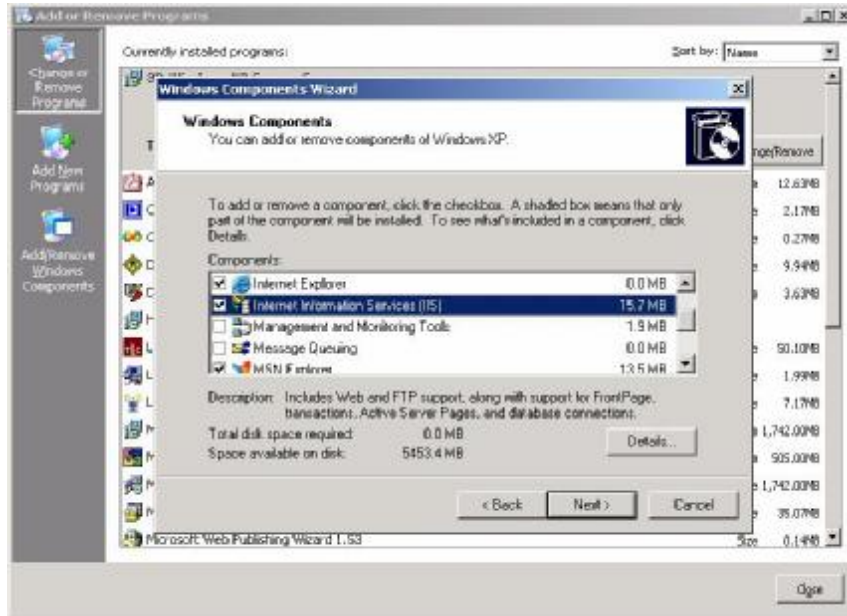
هذا الكتاب مختص ببرمجة Visual Basic .NET، والتحدث عن تطبيقات او برامج اخرى خارج نطاق الكتاب، الا ان هذا القسم موجه الى أولئك الأشخاص الذين ليس لديهم خلفية باستخدام الخادم IIS، ويوضح لهم باختصار شديد - طريقة تركيبه وعمله، حيث انه يعتبر احد المتطلبات الأساسية لتشغيل تطبيقات ASP.NET على الجهاز.

تركيب الخادم IIS

حتى تتمكن من تشغيل تطبيقات ASP.NET لابد ان يكون جهازك خادم ويب Web Server، وحتى تتمكن من تحويل جهازك الى خادم Web Server، عليك القيام بتركيب الخادم Internet Information Server من شركة Microsoft.

يمكنك تركيب انواع اخرى من الخوادم المنتشرة في الأسواق، ولكن على حسب علمي - حتى لحظة كتابة هذا الفصل - يعتبر الخادم IIS هو الخادم الوحيد الذي يمكنك من تشغيل صفحات ويب والمبنية على تقنية ASP.NET.

ان كنت تستخدم الإصدار Windows 2000 وما بعده، فيسرنى إخبارك بان الخادم IIS موجود ضمن الاسطوانة الأصلية لنسخة نظام التشغيل Windows. وكل ما هو مطلوب منك لتركيب الخادم IIS، هو تشغيل الرمز Add or Remove Programs الموجود في لوحة التحكم Control Panel، ومن ثم الانتقال الى خانة التثبيت Add/Remove Windows Components، واختيار العنصر Internet Information Services Components في يسار صندوق الحوار، واختيار العنصر (شكل 1-20).



شكل 1-20: تركيب الخادم IIS.

بعد تثبيتك للخادم IIS، قد لا يطلب منك برنامج الإعداد إعادة تشغيل الجهاز، مع ذلك من المستحسن دائماً إعادة تشغيله، وبمجرد بدء اقلاع نظام التشغيل فقد تحول جهازك الشخصي الى خادم ويب Web Server يمكن له ان يستضيف مواقع في أقراصه الصلبه لو كنت تملك بنية اتصال قوية.

وحتى نتأكد من نجاح عملية تركيب الخادم، قم بتشغيل المتصفح Internet Explorer، واكتب هذا الرابط:

`http://localhost`

ان تم فتح الصفحة بنجاح، فهذا يعني ان عملية تركيب الخادم IIS قد تمت بنجاح وتحول جهازك الى خادم ويب Web Server. اما ان لم تظهر لك نتائج ايجابية، حاول مراجعة مكتب الدعم الفني لشركة Microsoft في منطقتك.

ملاحظة

قد لا تكون متصل بشبكة محلية Local Network او طلب هاتفي Dial-up Networking، لذلك يفضل تحديد الاختيار Bypass proxy server for local addresses والخاصة باعدادات البروكسي التابعة للمتصفح Internet Explorer.

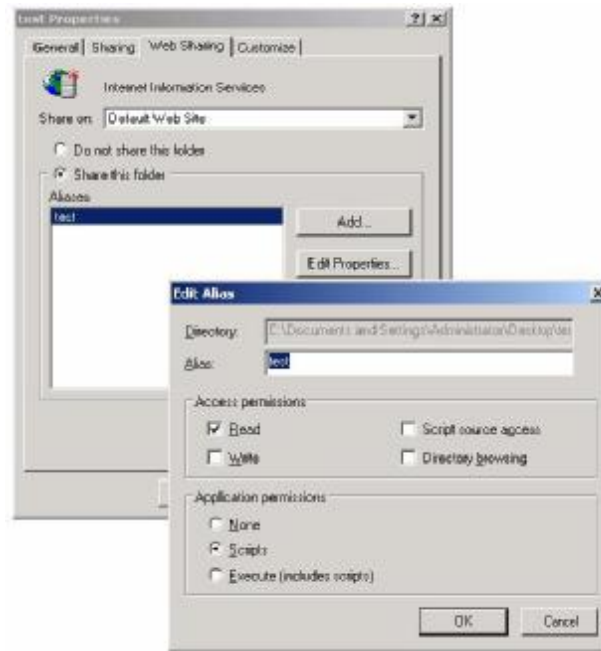
الأدلة الوهمية

الموقع الذي تريد استضافته ما هو الى مجموعة مترابطة من الملفات، وكما تفعل مع ملفاتك الشخصية بترتيبها وتوزيعها على مجلدات، فعليك توزيع ملفات المواقع على مجلدات، ولكن السؤال الذي يطرح نفسه، كيف يمكن لزائر موقعك او انت من الوصول الى المجلد الذي وضعت الملفات فيه، والإجابة عن طريق كتابة المجلد في عنوان الرابط URL.

ولكن المشكلة تكون ان الزائر لا يعلم شيئاً عن الموقع الفيزيائي للمجلد ومساره الكامل، وذلك لان الزائر لا يصل الى ملفات كما تصل انت، لذلك عليك إنشاء دليل وهمي Virtual Directory ما هو الا اسم وهمي مستعار يتم إيصالك الى المجلد الحقيقي.

حتى تقوم بعمل ذلك، أنشئ مجلد Folder كما تفعل دائماً، وانقر بزر الفأرة الأيمن على المجلد ثم اختر الامر Properties من القائمة المنبثقة، انتقل الى خانة التتويب Web Sharing ثم

حدد الاختيار Share this folder، وان لم يظهر لك صندوق حوار جديد فاضغط على الزر Add (شكل 20-2).



شكل 20-2: إنشاء دليل وهمي.

في خانة Alias اكتب الاسم الوهمي والمستعار الذي يمكن الزائر من الوصول الى المجلد بكتابته عند رابط URL، فلو كان الاسم الوهمي test يمكنك الوصول الى ملفات المجلد بكتابة شيئاً مثل:

`http://localhost/test`

وحتى تتأكد من ذلك، أنشئ ملف `test.html` واكتب فيه ما تشاء، ثم انسخه في نفس المجلد الذي أنشأته للتو، واكتب العنوان التالي في نافذة المتصفح:

`http://localhost/test/test.html`

ستلاحظ ان صفحتك التي صممتها للتو قد ظهرت في المتصفح.

ملاحظة

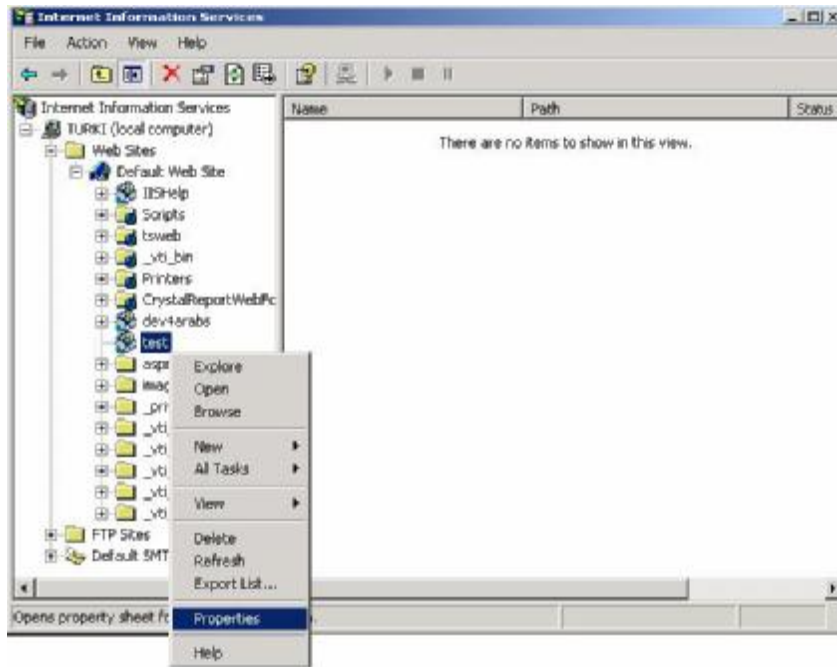
ان تجاهلت كتابة اسم الملف، سيقوم الخادم IIS بالبحث عن مجموعة ملفات افتراضية ك Default.html، Index.html، ...الخ.

المزيد ايضا، ان كنت متصلا بشبكة الانترنت، فقد تسمح للزوار بدخول موقعك ان علموا برقم المعرف **IP Address**، ليصلوا إليك بكتابة شيئاً مثل:

`http://999.999.999.999/test/test.html`

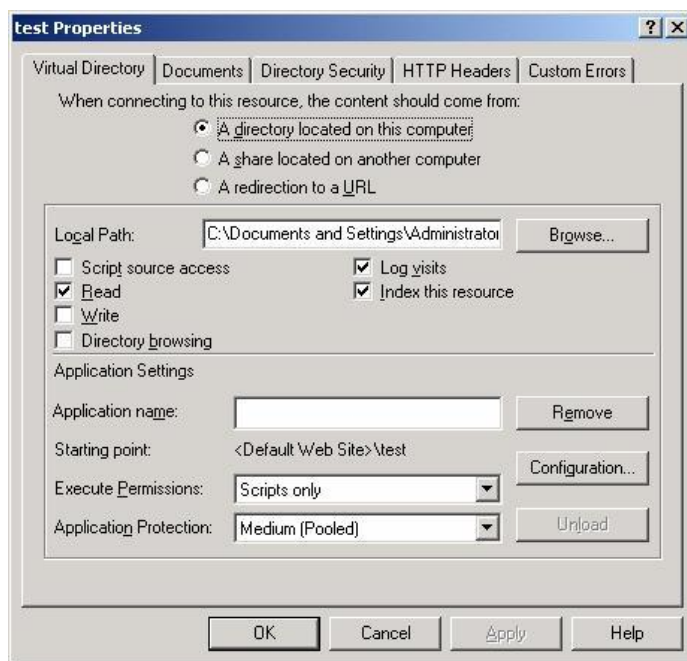
الوصول إلى الأدلة الوهمية

يمكنك الوصول الى كافة الأدلة الوهمية والمعرفة في الخادم IIS باختيار الرمز Internet Information Services من المجموعة Administrative Tools في لوحة التحكم Control Panel (شكل 20-3).



شكل 20-3: الأدلة الوهمية في الخادم IIS.

المزيد ايضا، تستطيع اضافة المزيد من الأدلة الوهمية من نفس النافذة، كما يمكنك حذفها ايضا، وان اردت تعديل خصائص الأدلة الوهمية فاضغط بزر الفارة الأيمن على عنصر الدليل الوهمي في الشجرة اليسرى، واختر الأمر Properties ليظهر لك الخادم IIS عشرات الخيارات والأوامر التي يمكنك تعديلها (شكل 20-4)، يمكنك مراجعة ملفات التعليمات والخاصة بالخادم IIS لمزيد من التفاصيل.



شكل 20-4: تعديل خصائص الدليل الوهمي.

مدخلك إلى نماذج Web Forms

عندما تتوى تطوير تطبيقات ASP.NET، فأنك من البديهي لن تظهر نوافذ لنماذج Windows Forms، وانما ستعتمد على نماذج أخرى شبيهة بها الى حد كبير - تسمى نماذج Web Forms. تمثل الصفحات التي تود عرضها على المستخدم في المتصفح Browser. ان كان الغرض من نماذج Windows Forms هو عرض واجهة الاستخدام للمستخدم على شكل نوافذ، فان نماذج Web Forms هدفها هو تحويل المخرجات الى شيفرات HTML ليتم عرضها على المتصفح.

تتميز نماذج Web Forms عن نماذج Windows Forms بعدم ضرورة توفر نسخة من إطار عمل .NET Framework. في جهاز الزائر، بل حتى لا يشترط وجود نظام تشغيل من نظم تشغيل Windows، ولكن يعيها انها اقل إمكانيات ومميزات من نماذج Windows Form - بشكل مبدئي.

عندما يود المستخدم الوصول الى صفحة من صفحات موقعك، فسيقوم بكتابة اسم الموقع الكامل ويشمل ملف الصفحة والذي يكون بالامتداد .aspx، والذي يمثل صفحة نموذج Web form page، فلو كان تطبيقك سيعرض 10 نماذج Web forms، فستحتاج الى 10 ملفات .aspx* بشكل مبدئي، كما يمكنك الربط بين الملفات المتعددة -كأن تخصص ملف لرأس جميع صفحات موقع Header أو أسفل جميع الصفحات Footer- كما سنرى لاحقا.

إنشاء المشروع

يمكنك الاعتماد على المفكرة Notepad ان كنت تنوي تطوير تطبيقات ASP.NET، كما هو الحال ما سائر تطبيقات .NET. الاخرى (Windows Applications، Console Application... الخ)، الا انني لا اجد سببا مقنعا يمكنك من الاستفادة من بيئة Visual Studio .NET والتي توفر لك أدوات في قمة الروعة لتسهيل حياتك البرمجية. اختر الامر New->Project من قائمة File، ليظهر لك صندوق حوار مشروع جديد New Project (شكل 20-5)، حدد القالب ASP.NET Web Application، ستلاحظ ان خانة اسم المشروع Name معتمدة ولا يمكن تعديلها فهي تحمل اسم مسار المشروع، ستلاحظ ان خانة مسار الملفات Location تطلب منك رابط URL للموقع الذي تريد تحميل الملفات اليه (الدليل الوهمي Virtual Directory)، اصف مثلا <http://localhost/test>، كما يمكنك كتابة اسم مسار المجلد -عوضا عن موقعه- بالضغط على الزر Browse.

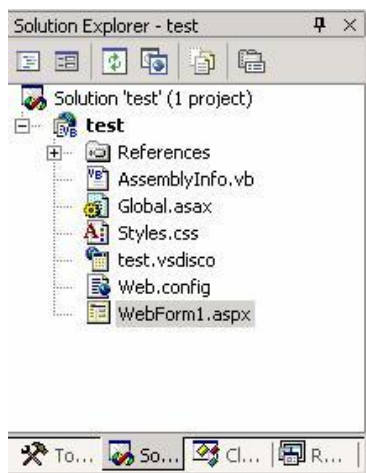


شكل 20-5: تحديد رابط الموقع المحلي الذي تود تحميل الملفات اليه.

ملاحظة

ان كنت قد وضعت حالة المتصفح Internet Explorer الى حالة العمل دون اتصال Work Offline، فلن يتم إنشاء ملفات المشروع بشكل صحيح وستظهر رسالة خطأ، لذلك عليك الغاء هذه الحالة بتشغيل المتصفح Internet Explorer نفسه، وإلغاء الاختيار Work Offline من قائمة File.

بعد ضغطك للزر OK، ستبدأ بيئة التطوير Visual Studio .NET بفتح مشروع جديد يحتوي على مجموعة من الملفات (شكل 20-6)، سنتعرض الى هذه الملفات بالتفصيل لاحقاً، رغم ان بعض الملفات كـ AssemblyInfo.vb ليست غريبة عليك ان كنت قد طورت مشاريع Windows Application، وبالنسبة للملف Styles.css فهو ملف اختياري تكتب فيه أنماط Cascading Style Sheet، اما الملف WebForm1.aspx فهو ملف صفحة نموذج Web Form التي تصممها وتود عرضها على المستخدم في المتصفح.

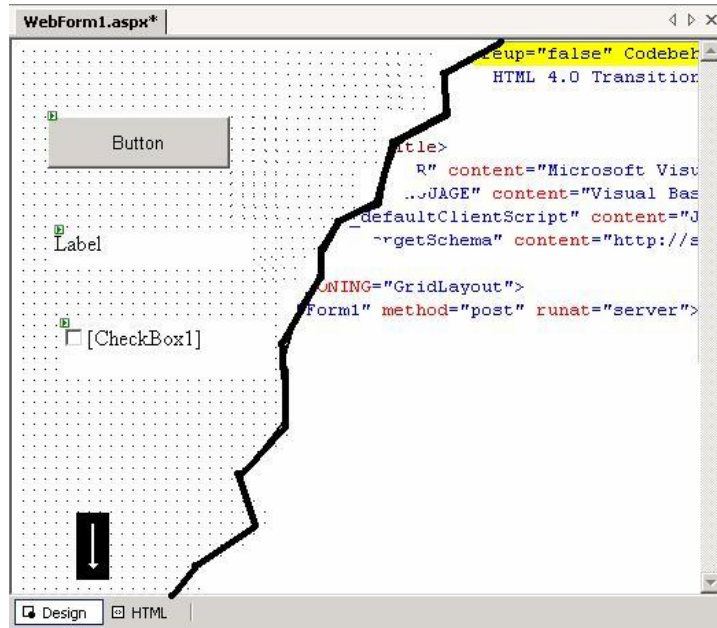


شكل 20-6: ملفات المشروع الابتدائية في نافذة مستكشف الحل Solution Explorer.

ملاحظة

انماط Cascading Style Sheet (CSS) غرضها تعريف انماط لتنسيقات ثابتة تستخدمها لعرض وسوم صفحات HTML. تمتلئ مواقع الانترنت بدروس ومقالات حولها، كما يمكنك العودة لمكتبة MSDN للحصول على مرجع كامل لجميع التنسيقات للانماط التي تعرفها.

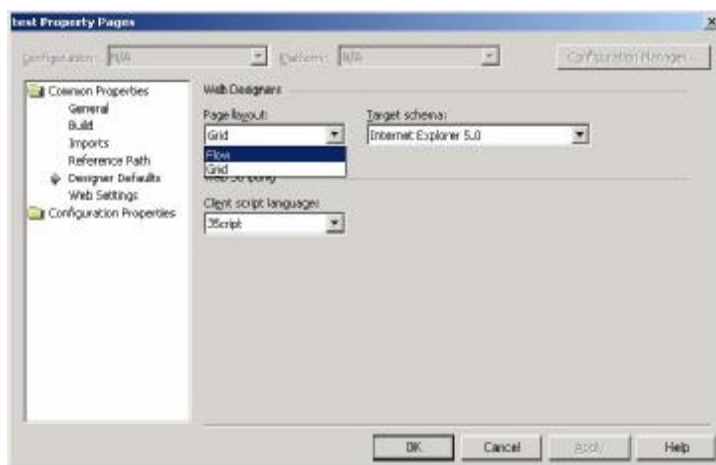
وكما هو الحال ما اغلب برامج تصميم المواقع، يمكنك عرض صفحة النموذج Web Form في الوضع Design او الوضع HTML، في الوضع الاول يمكنك تصميم صفحة النموذج بشكل مرئي Visual كما يمكنك اضافة الادوات وتعديل خصائصها، اما في وضع HTML فانك ستقوم بتصميم الصفحة بكتابة وسوم HTML يدويا بنفسك، يمكن التبديل بين كلا الوضع عن طريق الضغط على الزر المناسب في الزاوية اليسرى السفلية من نافذة المصمم (شكل 20-7).



شكل 20-7: التحويل بين عرض التصميم Design او شيفرات HTML.

ضبط الاعدادات الرئيسية

قبل البدء بوضع الأدوات على صفحات النماذج وكتابة الشيفرات، من المهم جدا ضبط اعدادات ثلاث خصائص هامة تؤثر في الشيفرة التي ولدها المصمم هي Page Layout، Target Schema، و Client Script Language، يمكنك الوصول لهذه الاعدادات في خانة التثبيت Designer Defaults الموجودة في صندوق حوار خصائص المشروع Project Property Pages (شكل 20-8).



شكل 20-8: ضبط الاعدادات الرئيسية للمصمم.

في هذه الاعدادات تخبر المصمم الطريقة المثلى لتحويل تصاميمك الى وسوم HTML، في الخانة Page Layout تحدد فيها نسق وضع الادوات على صفحات النماذج وهو اما يكون Flow او Grid، في الحالة الاولى يتم وضع الادوات على صفحات النماذج بحيث توافق اسلوب HTML لعرضها، وهو الحال كما تفعل مع برامج معالجة النصوص، وضع في عين الاعتبار ان اي تحجيم في نافذة المتصفح سيؤدي الى زحزحة الادوات وتغيير مواقعها ما لم تضبط الخصائص المناسبة لحجمها، اما في حالة Grid فسيتم وضع الادوات كما تفعل مع نماذج Windows Forms وستعرض الادوات على المتصفح تماما مثل تصميمها بمصمم صفحات النماذج Web Forms.

بالنسبة للاختيار Target Schema ففيه تحدد نوع المتصفح الذي ستعرض صفحات موقعك عليه، الميزة في اختيارك اصدارات قديمة لـ Netscape و Internet Explorer 3.03 Navigator 3 سيؤدي الى توافق وتطابق حقيقي مع اغلب المتصفحات المستخدم، الا ان العيب فيها هو عدم الاستفادة من الإمكانيات المتقدمة التي يوفرها الإصدار الأحدث Internet Explorer 5 وما بعده.

بعض شيفراتك البرمجية والتي يفترض ان تعمل في جهاز العميل Client (اي زائر الصفحة)، سيتم تحويلها الى لغة JScript، والمتوافقة مع جميع المتصفحات، اما ان كنت تتوقع عرض موقعك في متصفحات Internet Explorer فقط (كان يعرض في شبكة محلية)، فيمكن تغيير الاختيار Client Script Language الى VBScript.

كتابة الشيفرات

التعامل مع صفحة النموذج Web Form شبيه الى حد كبير مع التعامل مع نافذة النموذج Windows Form، وحتى نرى الامر واقعا اضف اداة Label في اعلى صفحة النموذج، واسفلها زر Button يليه اداة TextBox، وانقر نقرا مزدوجا على الزر لتتمكن من فتح نافذة الشيفرة واكتب الامر السطر التالي في حدثه Click:



```
Private Sub Button1_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles Button1.Click

    Label1.Text = "مرحبا بك يا " & TextBox1.Text
End Sub
```

مبروك! لقد قمت بالانتهاء من تصميم اول تطبيق لك مبني على تقنية ASP.NET، ان لم تصدق كلامي جرب الضغط على المفتاح [F5] لتنفيذ البرنامج، سيتم فتح المتصفح Internet Explorer الى الرابط <http://localhost/xxx/yyy.aspx> (حيث تمثل الحروف xxx الدليل الوهمي لمشروعك، والحروف yyy اسم ملف صفحة النموذج Web Page)، ويعرض لك صفحة النموذج التي صممتها للتو، جرب كتابة اسمك ثم اضغط على الزر Button، ستلاحظ ان عملية طلب Request جديدة تمت للصفحة وعادت بنفس الصفحة السابقة ولكن بعد تنفيذ الشيفرة السابقة لتظهر الاسم في الاداة Label.



شكل 20-9: بعد الضغط على الزر Button فتحت صفحة جديدة ظاهرة النص في الاداة Label.

نقطة هامة جدا جدا جدا: الشيفرة البرمجية تم تنفيذها على الخادم Server وليس العميل Client!

الشائب رقم Q315158 عند تنفيذ:

اعلنت شركة Microsoft قبل فترة عن ظهور شائب (تجده في المقال رقم Q315158 لموقع MSDN) ظهر في معظم اجهزة المستخدمين، وبالتحديد الذي يودون تجربة تنفيذ تطبيقات ASP.NET على اجهزتهم الشخصية تحت الخادم IIS، وذلك تظهر لهم رسالة (شكل 20-10) عند تنفيذ شيفرة ASP.NET التي أنجزوها.



شكل 20-10: شائب عند تنفيذ صفحة ASP.NET.

ان كنت تود معرفة تفاصيل سبب حدوث هذا الخطأ، فيمكنك توجيه متصفحك الى الرابط <http://support.microsoft.com/default.aspx?scid=kb:en-us:Q315158> حيث اني لن اعرض لك هنا الا حل من الحلول الثلاثة التي يعرضها الرابط السابق.

افتح ملف الاعدادات Machine.config وابحث عن العنصر processModel، وستجد بين ثانياً مواصفاته المواصفة userName Attribute:

```
<processModel enable="true"
...
...
userName="machine" password="AutoGenerate" logLevel="Errors"
...
...
/>
```

غير قيمة هذه المواصفة من machine الى SYSTEM:

```
userName="SYSTEM" password="AutoGenerate" logLevel="Errors"
```

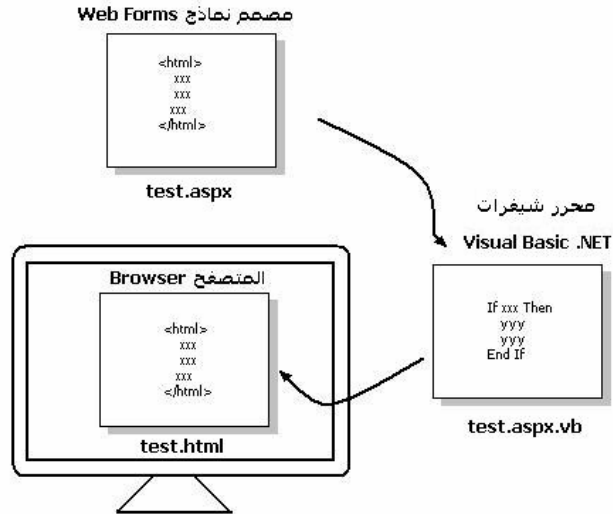
احفظ التعديلات، واعد تنفيذ مشروعك لتستمتع ببرمجة ASP.NET.

انظر ايضا

لمزيد من التفاصيل حول ملفات الاعدادات Configuration Files، راجع الفصل الحادي عشر **المجمعات Assemblies**.

تحليل الشيفرة

تقتضي فلسفة تطبيقات ASP.NET والمبنية على نماذج Web Forms فصل الشيفرات البرمجية عن الشيفرات المخصصة لعرض واجهة الاستخدام، وحتى تفهم ماذا وكيف ولماذا يحدث ذاك وذلك، عليك معرفة ان كلمة الشيفرة -التي اقصدها في هذا السياق- مقسمة إلى ثلاثة أقسام (شكل 20-11).



شكل 20-11: الأقسام الثلاث للشيفرات.

اول شيفرة تم توليدها هي شيفرة HTML من مصمم نماذج Web Forms، بحيث تحتوي على بيانات النموذج والادوات المحضونة فيه، والشيفرة التي تليها هي شيفرتك المكتوبة بلغة Visual Basic .NET والتي تقوم بمعالجة الطلبات Requests من الزوار واجراء اللازم، لتنتج الشيفرة النهائية HTML والتي يتم عرضها على المتصفح Browser، واليك تفاصيلها:

شيفرة HTML مولدة من مصمم النماذج:

بعد وضع الادوات على صفحة النموذج Web Form، سيقوم مصمم النماذج بتوليد الشيفرة التالية:

```

<%@ Page Language="vb" AutoEventWireup="false"
Codebehind="WebForm1.aspx.vb" Inherits="test.WebForm1"%>
<HTML dir="rtl">
  <HEAD>
    <title>WebForm1</title>
    ...
  </HEAD>
  <body MS_POSITIONING="GridLayout">
    <form id="Form1" method="post" runat="server">

```

```
<asp:Label id="Label1" runat="server"></asp:Label>
<asp:Button id="Button1" runat="server"></asp:Button>
<asp:TextBox id="TextBox1" runat="server"></asp:TextBox>
</form>
</body>
</HTML>
```

ضع في عين الاعتبار، ان الشيفرة السابقة شبيهة الى حد كبير بشيفرة HTML ولكن لا يوجد متصفح يستطيع فهمها، وعندما تمنع النظر بين وسومها ستلاحظ الموصفتين ID و runat، الاولى تمثل اسم الاداة التي وضعتها، والثانية تحدد فيها اين يتم معالجة شيفرة الاداة، والقيمة "server" المسندة لها تؤكد لنا على ان جميع شيفرات وأوامر الأدوات يتم تنفيذها في الخادم Server وليس جهاز العميل Client.

الشيفرة السابق ستحفظ في نفس ملف مصمم النماذج Web Form بالامتداد .aspx.

شيفرة Visual Basic .NET:

عندما يقوم الزائر بطلب الصفحة وكتابة اسمها في المتصفح، سيتم تنفيذ الشيفرة التي كتبناها بلغة Visual Basic .NET:

```
Private Sub Button1_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles Button1.Click

    Label1.Text = "مرحبا بك يا " & TextBox1.Text
End Sub
```

والتي تقوم بمعالجة البيانات واسناد قيمة النص الموجود في الاداة TextBox إلى الاداة Label، الشيفرة السابقة -كما قلت للمرة الثالثة- سيتم تنفيذها على جهاز الخادم لمعالجة الطلبات من العملاء Clients، فلا تعرض فيها نوافذ نماذج Windows Forms او صناديق حوار Message Box او اي مخرجات على الشاشة، والسبب ان المخرجات يفترض ان تكون بصيغة HTML لترسل الى جهاز العميل Client وتعرض في داخل المتصفح Browser.

هذه الشيفرة ستحفظ في ملف يحمل نفس اسم ملف صفحة النموذج السابقة ولكن بالامتداد .aspx.vb، فلو كان اسم الملف النموذج test.aspx فسيكون اسم هذا الملف test.aspx.vb .

شيفرة HTML النهائية:

اخيراً، بعد قيام الخادم بمعالجة الطلب Request، سيتم توليد شيفرة HTML النهائية والتي يمكن للمتصفح Browser من استيعابها وعرضها بتنسيقات مختلفة أمام انف الزائر:

```
<HTML dir="rtl">
  <HEAD>
    <title>WebForm1</title>
  </HEAD>
  <body MS_POSITIONING="GridLayout">
    <form name="Form1" method="post" action="WebForm1.aspx"
id="Form1">
      ...
      ...
      <span id="Label1" style="font-size:Large;font-
weight:bold;height:72px;width:335px;Z-INDEX: 101; LEFT: 92px;
POSITION: absolute; TOP: 48px">مرحباً بك يا عباس السريع</span>
      ...
      ...
    </form>
  </body>
</HTML>
```

أساليب تنفيذ الصفحة

وضحت في الفقرة السابقة توزيع الشيفرات عند طلب الزائر لصفحة aspx من صفحات موقعك، وسأظهر لك هنا المزيد من التفاصيل حول الأساليب الثلاث التي يمكن معالجة صفحة ASP.NET بها، ولكن دعني اتفق معك اني في كل مرة اذكر فيها عبارة **موجه الصفحة @Page**، فإنني اقصد هذا السطر الموجود في اعلى ملف صفحة النموذج aspx:

```
<%@Page Language="vb" AutoEventWireup="false"
Codebehind="test.aspx.vb" Inherits="test.WebForm1"%>
```

فئات Code-Behind Classes:

عندما تقوم بتنفيذ برنامجك، ستقوم بيئة التطوير Visual Studio .NET بترجمة ملفات مشروعك المكتوبة بلغة .NET Visual Basic (والتي تنتهي بالامتداد *.aspx.vb) الى مكتبة بالامتداد DLL، وبمجرد قيام الزائر بطلب صفحة نموذج aspx، سيقوم محرك ASP.NET بقراءة القيمة Inherits في موجه الصفحة @Page، القيمة تمثل الاسم الكامل لفئة الصفحة التي كتبت بها شيفرات الأدوات -كما فعلنا سابقاً مع الاداة Button، وتنفيذ الإجراء المناسب في فئة الصفحة.

ملفاتك المكتوبة بلغة .NET Visual Basic والتي تم ترجمتها تسمى الشيفرة الخلفية -Code-Behind.

الترجمة عند الطلب On-Demand Compilation:

لست مضطرا لترجمة ملفات الشيفرة الخلفية في مكتبة DLL، إذ يمكنك استخدام الموصفة src في موجه الصفحة @Page وكتابة اسم الملف المصدري للشيفرة الخلفية:

```
<%@ Page Language="vb" src="WebForm1.aspx.vb" _
Inherits="test.WebForm1"%>
```

عندما يقوم الزائر بطلب صفحة aspx، سيتم البحث عن الملف الموجود في الموصفة src ويتم ترجمته إلى ملف DLL عند الطلب، يعيب هذا الأسلوب البطء الشديد عند أول استدعاء للصفحة، كما أن شيفراتك المصدرية ستكون قابلة للتعديل فهي مازالت محفوظة في ملفات النصية *.aspx.vb.

الأسلوب الكلاسيكي Classic ASP Style:

الأسلوب الكلاسيكي يتبع أسلوب استخدام تقنية ASP القديمة في كتابة الصفحات، يعيب هذا الأسلوب مثل ما يعيب تقنية ASP القديمة حيث تدمج شيفرات واجهة الاستخدام، بنفس شيفرات لغة Visual Basic .NET، مما يعقد تركيبة ملفات aspx -خاصة أن كبر حجمها. على أية حال، يمكنك اتباع الأسلوب الكلاسيكي بوضع الشيفرة التي تنوي تنفيذها على الخادم داخل التركيب <SCRIPT RUNAT="server"> ... </SCRIPT>:

```
<%@ Page Language="vb"%>
...
...
...

<SCRIPT RUNAT="server">
    Private Sub Button1_Click(ByVal sender As System.Object, _
        ByVal e As System.EventArgs) Handles Button1.Click

        Labell1.Text = " مرحبا بك يا " & TextBox1.Text
    End Sub
</SCRIPT>
```

الخلاصة

نستنتج من كل عرضناه في الفقرات السابقة، ان فلسفة ASP.NET تقتضي ربط الادوات التي تعرض عند المستخدم في صفحة مستعرضه، بشيفرات برمجية تكتب بين فكي احداث هذه الادوات. ان لم نبرمج سابقا بأي لغة برمجة ويب لجهة الخادم Server side script (كـ ASP، CGI، Perl، PHP، ... الخ)، فعليك تغيير منطقك البرمجي ليميل الى جانب الخادم وليس العميل، وان لم تضع ذلك في عين الاعتبار دائما، ستواجه الكثير من المتاعب.

الفئة Page

ان كانت الفئة System.Windows.Forms.Form تمثل نافذة نموذج Windows Form، فان الفئة System.Web.UI.Page تمثل صفحة نموذج Web Form:

```
Public Class WebForm1
    Inherits System.Web.UI.Page
    ...
    ...
    ...
End Class
```

مع ذلك الادوات وصفحات النماذج ليست كالأدوات ونوافذ النماذج التابعة لـ Windows Forms، حيث تحتوي الثانية على عدد اكثر بكثير من الخصائص، الطرق، والاحداث -كما ستكتشفه بنفسك.

في هذا القسم سأحدث عن الفئة Page وسأحاول التعرض الى مجموعة من خصائصها، طرقها، واحداثها. كما سأذكر مجموعة اضافية من الوسوم التي تستخدمها بالاضافة الى موجه الصفحة @Page.

خصائص صفحة النموذج

عندما ننوي انشاء صفحة نموذج عربية، فأول خاصية عليك تغيير قيمتها الخاصةية dir والتي تسند لها القيمة rtl لتحويل اتجاه الصفحة من اليمين الى اليسار، والخاصية Culture التي تحدد الاعدادات الإقليمية Regional Settings لصفحة النموذج.

عند حدوث خطأ ووقوع استثناء Exception اثناء تنفيذ شيفرة الصفحة ولم تتفاداه، سيقوم محرك ASP.NET بعرض صفحة خاصة به توضح الاستثناء ومصدره، مع ذلك قد تعين صفحة خطأ خاصة بموقعك وذلك بإسناد رابط الصفحة الى الخاصية `ErrorPage`:

```
me.ErrorPage = "http://www.dev4arabs.com/error.aspx"
```

تفيدك الخاصية `IsPostBack` لمعرفة ما اذا كانت هذه اول مرة تعرض فيها الصفحة، او تم عرضها بعد ارسال البيانات لها -اي قام الزائر بالضغط على زر `Submit`، تعود هذه الخاصية بالقيمة `True` ان تم فعلا ارسال البيانات الى هذه الصفحة. بالنسبة للخاصية `Controls` فه تمثل مجموعة للادوات المحضونة في صفحة النموذج، يمكنك تطبيق الحلقة `Next ... For Each` عليها بكفاءة -كما فعلنا سابقا مع نوافذ النماذج:

```
Dim ctl As Control
For Each ctl In Me.Controls
    ...
Next
```

معظم الخصائص الاخرى (على وجه الخصوص `Request`، `Response`، `Server`، `Application`، `Session`) تعود بكائنات من النوع `HttpRequest`، `HttpResponse`، `HttpSessionState`، `HttpApplicationState`، `HttpServerUtility` سأحدث عنها في الفصل القادم **تطبيقات ASP.NET (2)** بمشيئة الله.

الخاصية `EnableViewState` والخاصية `ViewState`:

عندما يقوم الزائر بزيارة الصفحة، فان القيم التي تحملها الادوات ستكون هي القيم الافتراضية - والتي عرفت لحظة تصميم صفحة النموذج، مع ذلك قد تود بالاحتفاظ بالتعديلات التي قام بها المستخدم، تخيل مثلا انك قمت بوضع مجموعة من الادوات `TextBox` تطلب من المستخدم ادخال معلومات، وافترض ان المستخدم قام بتعبئتها جميعا وضغط على الزر `Submit`، وتخيل ان احد المعلومات التي ادخلها خاطئة، لذلك يتوجب عليك إظهار ادوات `TextBox` له من جديد حتى يصحح تعديلاته.

المشكلة تظهر بعد هذه الفاصلة، عندما يتم عرض الادوات للمرة الثانية فان القيم التي ادخلها المستخدم في الزيارة الاولى للصفحة (قبل الضغط على زر Submit) سيتم إغائها وكان شيئاً لم يحدث، مما يحد المستخدم الى اعادة كتابة جميع البيانات من جديد.

من هنا يأتي دور الخاصية `EnableViewState` والتي تسند لها القيمة `True` لتمتلك من حفظ القيم الحالية للادوات لحظة ارسالها للمرة الثانية الى نفس الصفحة:

```
Me.EnableViewState = True
```

ضع في عين الاعتبار، ان جميع الادوات سيتم حفظ قيمها ان كانت قيمة الخاصية `EnableViewState` لكل اداة هي `True`، وهذا بحد ذاته سيكلفك الكثير من البيانات المرسلة في كل مرة للصفحة، لذلك لا تستخدم هذه الخاصية الا ان كنت فعلاً بحاجة لها، واسند القيمة `False` دائماً للادوات الغير مرغوبة حفظ قيمها -خاصة التي تحتوي على قيم كثيرة:

```
Listbox1.EnableViewState = False
```

بمجرد اسنادك للقيمة `True` للخاصية `EnableViewState` سيتم حفظ قيم الادوات، ولكن ما هي علاقة الخاصية `ViewState` المسطورة كعنوان لهذه الفقرة؟ والاجابة ببساطة هو إمكانية تعريفك لمتغيرات او قيم -ان صح التعبير - يمكنك الاحتفاظ بها حتى تستفيد منها للاستدعاء الثاني للصفحة، الشيفرة التالية مثلاً تعرض لك كم مرة تم استدعاء الصفحة:

```
If Me.ViewState("Counter") Is Nothing Then
    Me.ViewState("Counter") = CInt(1)
Else
    Me.ViewState("Counter") = CInt(Me.ViewState("Counter") + 1)
End If
Label1.Text = CStr(Me.ViewState("Counter"))
```

ملاحظة

الخاصية `ViewState` هي كائن من النوع `Dictionary` -وهو شبيه الى حد ما بالمجموعات `Collection`، وقد تمكنا من تعبئة عناصره دون استخدام الطرق التقليدية للمجموعات `Add()`، `Insert()`... الخ. لم اتحدث في هذا الكتاب عن الكائن `Dictionary`، لذلك أنصحك بمكتبة MSDN ان اردت المزيد من التفاصيل حول هذا الكائن.

الخاصية SmartNavigation:

عندما يقوم الزائر بالضغط على الزر Submit، ستعود نفس الصفحة اليه في المستعرض، وقد يلاحظ المستخدم التغيير في نزول النسخة الجديدة من الصفحة مما يسبب التشويش عليه، كما ان موقع اشربة التمرير ScrollBars للمتصفح ستبدأ من اعلى الصفحة عند انزال النسخة الجديد، مع ذلك يمكنك اسناد القيمة True للخاصية SmartNavigation لمنع حدوث كل ذلك.

ملاحظة

مفعول الخاصية SmartNavigation سيظهر ان كان المتصفح الإصدار الخامس من Internet Explorer وما بعده.

طرق صفحة النموذج

من طرق صفحة النموذج الطريقة MapPath() والتي تحول مسار الدليل الوهمي Virtual Directory الى المسار الفيزيائي (الحقيقي) له:

```
Dim F As New System.IO.StreamReader(Me.MapPath("/test/file.txt"))
```

كما توجد الطريقة -أشبه بخاصية- هي HasControls() والتي تعود بالقيمة True ان وجدت ادوات على صفحة النموذج.

أحداث صفحة النموذج

عندما يتم طلب صفحة نموذج، فاول حدث تقوم بتنفيذه هو Init، وهو حدث ابتدائي يمثل عملية استدعاء للصفحة ولا يمكنك قراءة قيم الادوات التي كتبها المستخدم ولا حتى معرفة ما اذا كانت هذه المرة الاولى لعرض الصفحة، وذلك لان الحدث يتم تنفيذه قبل الربط الفعلي مع ادوات النموذج، فلا تعتمد على هذا الحدث كثيرا.

بعد ان يتم ربط الادوات بأحداثها وتحميل بياناتها، يمكنك كتابة الشيفرات في الحدث Load والذي يمكنك من قراءة قيم الادوات كما تستطيع الاستعلام عن الخاصية IsPostBack. وبمجرد تنفيذ جميع الشيفرات في هذا الحدث، سيتم تنفيذ احداث باقي الادوات في النموذج. الحدث الاخير الذي سيتم تنفيذه هو Unload والذي تكتب فيه جميع الشيفرات الضرورية لتفريغ المصادر كإغلاق اتصالات قواعد البيانات مثلا.

الغرض الوحيد للخاصية `ErrorPage` هو فقط عرض صفحة معينة ان حدث استثناء لم يتم تفاديه، اما إن اردت إجراء عمليات أخرى فيمكنك كتابة شيفرات هذه العمليات في الحدث `Error`.

وسوم إضافية

بعض الخصائص التي عرضتها عليك سابقا تكتب في داخل موجه الصفحة `@Page`، جرب مثلاً تغيير قيمة الخاصية `SmartNavigation` لتجد القيمة المسندة في موجه الصفحة `@Page` مكتوب:

```
<%@ Page Language="vb" ... .. smartNavigation="True"%>
```

يمكنك كتابة وسم صفحة `@Page` واحد فقط في كل صفحة ملف `aspx`، كما يفضل وضعه في أعلى الصفحة حتى تتمكن من الوصول إليه بشكل سريع. من الوسوم الإضافية الوسم `@Import` الذي يحاكي الكلمة المحجوزة `Imports` (في لغة `Visual Basic .NET`) لاستيراد مجال أسماء في نفس صفحة `aspx`:

```
<%@ Import namespace=" System.Data.OleDb"%>
```

والوسم `Implements` الذي يحاكي الكلمة المحجوزة `Implements` (في لغة `Visual Basic .NET`) والذي يضمن واجهة `Interface` في صفحة النموذج:

```
<%@ Implements Interface="System.IDisposable"%>
```

ملاحظة

إن أصبت بوسواس من الوسمين السابقين واعتقد أنك ستستخدمهما دائماً، فدعني اذكرك هنا بأنهما خاصان بصفحة النموذج `aspx` فقط، أما صفحة الشيفرة الخلفية `aspx.vb` فهي لا زالت بلغة `Visual Basic .NET` التي تعرفنا عليها منذ الفصل الأول لهذا الكتاب **تعرف على** `Visual Basic .NET`!

اخيراً، لديك الـ Reference الذي يمكنك من ربط صفحة نموذج أخرى في نفس الصفحة، للاستفادة من كائناتها (مع العلم انها لا تقوم بإظهارها وانما ربطها كمرجع كما تفعل مع صندوق حوار المراجع (Reference):

```
<%@ Reference page="WebForm2.aspx" %>
```

الأدوات

كما ذكرت في بداية القسم السابق، التعامل مع صفحات نماذج وادوات Web Forms شبيه الى حد كبير التعامل مع نماذج وادوات Windows Forms، ولكن عشرات الخصائص، الطرق، والاحداث ليست مدعومة في ادوات Web Forms، لدرجة ان بعض الادوات لا تحتوي على اية احداث!، والسبب -كما ذكرت- يتعلق بمكان تنفيذ الشيفرات المصدرية، فكيف تريد تنفيذ الحدث MouseMove على اداة Web Form ان كانت شيفراتها لا تنفذ الى في الخادم (لحظة ارسال البيانات الى الصفحة). ولو كان الامر كذلك، لثم تنفيذ الصفحة بالخادم والاتصال به في كل مرة تحرك الفأرة فوق الاداة من داخل متصفحك!

أدوات Web Forms Controls

هذه المجموعة من الادوات تجدها في خانة التويب Web Forms من صندوق الادوات Toolbox، ودعني اريك هنا بعض التغييرات التي تميزها عن ادوات Windows Forms. اول تغيير قد لاحظته ان خاصية الاسم هي (ID) وليس (Name)، وذلك يتعلق بصيغة لغة HTML لبناء المعرفات في وسومها، كما توجد خصائص اضافية كالخاصية BorderStyle التي تحدد فيها نقش الحد الخارجي للاداة، والخاصية BorderWidth لتذكر فيها عرض الحد. بالنسبة للخاصية cssClass ففي تذكر اسم فئة النمط Class Style والخاص بالانماط CSS التي تعرفها بنفسك. وضع في عين اعتبارك ان خاصية حجم الخط التابعة لكائن الخط Font لا تسند لها قيم عددية، اذ عليك استخدام الاحجام المعروفة (كـ Small، Big، Medium... الخ)، اما ان اردت استخدام قيم عددية، فلا تنسى اضافة الحرفين pt بعد الرقم (10pt، 20pt... الخ). معظم الخصائص قد تم اختصارها لبعض الادوات، فنجد مثلاً الاداة CheckBox والاداة RadioButton تحتوي على الخاصية Checked والتي تعود بالقيمة True ان تم اختيار العنصر، وعلى ذكر الاداة RadioButton فدعني أنبهك هنا بإمكانية تحديد اكثر من اداة

RadioButton في نفس الاداة الحاضرة! وحتى تمنع الزائر من عمل ذلك، اربط ادوات الـ RadioButton في مجموعات عن طريق كتابة اسم المجموعة في خاصيتها GroupName. ان كنت على دراية تامة بلغة تنسيق البيانات HTML، فالتعامل مع الادوات سيكون سهلا عليك، فكل ادا تراها لها مقابل في لغة HTML، كاداة الجداول Table تمثل الـ <table>، اداة Label تمثل الـ ، اداة الصور Image تمثل الـ ، اداة الرابط HyperLink تمثل الـ <a>، اداة Panel تمثل الـ <Panel>، اداة الـ ... علي تغيير عنوان الكتاب الى HTML ان اردت مني اكمال وشرح هذه الادوات.

أدوات HTML Forms Controls

الادوات السابقة تستخدمها ان اردت برمجتها وكتابة شيفرات مصدريه يتم تنفيذها في الخادم Server، اما ادوات HTML Forms Controls فالغرض منها اضافة ادوات في صفحة النموذج بحيث تكتب الشيفرات التي تود تنفيذها عند العمل (اي في داخل المتصفح). ولن تستطيع استخدام اي لغة من لغات NET. لبرمجة هذه الادوات، وليس لك مخرج إلا الاعتماد على احد لغات ويب للصفحات الديناميكية كـ VBScript، J Script، او حتى Java Script. يمكنك استخدام هذه الادوات من خانة التوييب HTML في صندوق الادوات Toolbox (10-20).



شكل 20-12: ادوات HTML Forms تصل إليها من خانة التبويب HTML في صندوق الادوات.

أدوات التحقق Validation

قد تلاحظ في خانة التبويب Web Forms من صندوق الأدوات مجموعة من الأدوات اسمها يحمل الصيغة xxxValidator، تستخدم هذه الأدوات في أغلب الأحوال - مع ادوات النص TextBox بحيث يمكنك من التحقق من البيانات قبل ارسالها الى الخادم، وعملية التحقق تتم بتوليد شيفرة مصدرية بلغة Jscript او VBScript بصفحة العميل في المتصفح. من هذه الادوات، الاداة RequiredFieldValidator للتحقق من انه تم ادخال قيمة في الاداة (التي ستكون اداة نص في اقصى الاحوال)، الاداة RangeValidator التي يمكنك من تحديد مجال للقيم يمكن ان تحمله الاداة، الاداة CompareValidator التي يمكنك من التحقق ما اذا كانت قيمة الاداة اكبر من، اصغر من، او تساوي قيمة اداة اخرى في نفس الصفحة، والاداة CustomValidator لتخصيص عمليات التحقق Validation بنفسك برمجيا (يمكن ان تكون شيفرة التحقق في الخادم ايضا عن طريق هذه الاداة).

ملاحظة

جميع هذه الأدوات (باستثناء RequiredFieldValidator) تجري عملية التحقق ان لم تكن القيمة خالية Empty، وان كانت خالية فسيعتبر التحقق ناجح. وتذكر انه سيتم تحويلها الى شيفرات Script لتعمل في المتصفح، لذلك قد يكون الزائر قد ألغى عمل تنفيذ شيفرات الـ Scripts في متصفحه ولن تعمل هذه الأدوات بنجاح.

الأدوات السابق ذكرها مشتق من الفئة القاعدية BaseValidator، واستخدامها متشابه الى حد كبير، ففي الخاصية ControlToValidate تحدد فيها اسم الأداة التي تود التحقق منها، ومن ثم تسند قيمة حرفية الى الخاصية ErrorMessage حتى تظهر رسالة الخطأ في وجه الزائر. وبمجرد قيام الزائر بالضغط على زر Submit، ستجرى عملية التحقق (شكل 20-13).



شكل 20-13: استخدام الأداة RequiredFieldValidator.

كان غرضي من هذا الفصل تعريفك بأساسيات تطوير تطبيقات ASP.NET Applications، وكان جل تركيزي على نماذج Web Forms وأدواتها. الفصل التالي سيعرض عليك مزيد من التفاصيل التي يتطرق بعضها حول نماذج Web Forms، والبعض الآخر حول مشاريع ASP.NET بشكل عام.

تطبيقات ASP.NET (2)

عرفتك في الفصل السابق على أساسيات تصميم تطبيقات ASP.NET وإنشاء صفحات نماذج Web Forms حتى تستوعب الفكرة من عمل صفحات ASP.NET، ولكن تبقى لنا مجموعة كبيرة من المواضيع المتفرقة التي عليك معرفتها حتى تنجز تطبيقات ASP.NET متكاملة.

كائنات صفحات ASP.NET الأساسية

الخصائص الخمس لصفحة النموذج Request، Response، Server، Application، و Session تعود بكائنات من النوع HttpRequest، HttpResponse، HttpServerUtility، و HttpSessionState، و HttpApplicationState -على التوالي، في هذا القسم سنتحدث عن هذه الكائنات بشكل سريع، ولمزيد من التفاصيل فان مستندات .NET Documentation بانتظارك.

الكائن HttpRequest

يمثل الكائن HttpRequest البيانات القادمة من متصفح العميل Client Browser، ويحتوي على مجموعة كبيرة من الطرق والخصائص -للقراءة فقط -ReadOnly- يمكنك من الحصول على معلومات حول العميل والبيانات التي أرسلت للصفحة. اول هذه المعلومات هي طريقة الحصول عليها ومعرفة كيف تم إرسالها عن طريق الخاصية HttpMethod، والتي تعود بقيمة حرفية من النوع String - تكون اما GET او :POST

```
Label1.Text = Me.Request.HttpMethod ' GET او POST
```

بالنسبة لإرسال البيانات بالأسلوب POST في أغلب الأحوال يتم بالضغط على الزر Submit في صفحة العميل، أما الأسلوب GET فتتم بكتابة الرابط URL مع إضافة القيم عليه:

```
http://www.dev4arabs.com/test.asp?id=1&site=vb
```

البيانات مرسلة في الرابط السابق بالأسلوب GET، وإن أردت قراءة القيم السابقة استخدم الخاصية `QueryString`:

```
Label1.Text = "ID & = " & Me.Request.QueryString("id")
Label 1.Text = "Site & = " & Me.Request.QueryString("site")
```

كما يمكنك تحديد نوع عملية إرسال البيانات (أما POST أو GET) عن طريق الخاصية `RequestType`:

```
Me.Request.RequestType = "POST"
```

المزيد أيضاً، تستطيع معرفة حجم البيانات المرسلة من صفحة العميل إلى الصفحة عن طريق الخاصية `ContentLength`:

```
Label1.Text = CInt(Me.Request.ContentLength)
```

أما عند رغبتك في الحصول على رقم معرف IP Address للزائر فاستعلم عنه في الخاصية `UserHostAddress`، وإن كان العميل يحتوي على اسم DNS فيمكن استخدام الخاصية `UserName`:

```
Label1.Text = Me.Request.UserHostAddress
Label2.Text = Me.Request.UserName
```

وعند الحديث عن المتصفح Browser، فيمكنك استخدام الخاصية `Browser` والتي تمثل كائن من النوع `HttpBrowserCapabilities` التي تحتوي على 25 خاصية تعود بمعلومات حول المتصفح، كاسم المتصفح، رقم إصداره، السماح ودعمه للغة VBScript، نظام التشغيل... إلخ:

```
Label1.Text = Me.Request.Browser.Browser      ' IE
Label2.Text = Me.Request.Browser.MajorVersion ' 6
Label3.Text = Me.Request.Browser.Platform     ' WinNT
Label4.Text = Me.Request.Browser.VBScript     ' True
```

اخيرا، ان كانت الصفحة ترسل لك ملف Upload، فيمكنك الوصول الى هذه الملفات والتحكم فيها عن طريق المجموعة Files، والتي تحتوي على كائنات من النوع HttpPostedFile، الشيفرة التالية تقوم بحفظ جميع الملفات المرسله الى الصفحة:

```
Dim uploadedFile As HttpPostedFile
For Each uploadedFile In Me.Request.Files
    uploadedFile.SaveAs(uploadedFile.FileName)
Next
```

الكائن HttpResponse

الكائن HttpResponse يمثل البيانات المرسله من الخادم الى صفحة العميل، يمكنك ارسال وسوم HTML ان اردت باستخدام الطريقة Write() لكتابة الوسوم كوسيطه لها، او استدعاء الطريقة WriteFile() ان كانت الوسوم محفوظة في ملف اخر:

```
Me.Response.Write ("<p><b>مرحبا بك</b></p>")
Me.Response.Write ("<p><b>مرحبا بك</b></p>")
Me.Response.WriteFile ("data.html")
```

من الطرق ايضا، الطريقة Redirect() التي توجه العميل الى صفحة اخرى، والطريقة End التي توقف عملية اكمال ارسال الصفحة:

```
Me.Response.Redirect("http://www.dev4arabs.com/accessDenied.aspx")
Me.Response.End()
```

عند التعامل مع الـ Cookies، فعليك إنشاء كائن من النوع HttpCookie تحدد في مشيده اسم الـ Cookie وقيمتها، وقد تعدل خاصيته Expires، ومن ثم تضيفه الى المجموعة Cookies التي تقبل كائنات من هذا النوع:

```
Dim userName As New HttpCookie("NAME", "تركبي العسيري")
Dim userPassword As New HttpCookie("PASSWORD", "1234")

userName.Expires = Date.Now.AddYears(1)
userPassword.Expires = Date.Now.AddYears(1)

Response.Cookies.Add(userName)
Response.Cookies.Add(userPassword)
```

اما عند القراءة، فلا تنسى ان البيانات المرسله من العميل الى الخادم تصطادها من الخاصية
Request وليس Response:

```
Label1.Text = Me.Request.Cookies("NAME").Value      ' تركي العسوي
Label2.Text = Me.Request.Cookies("PASSWORD").Value  ' 1234
```

الكائن HttpServerUtility

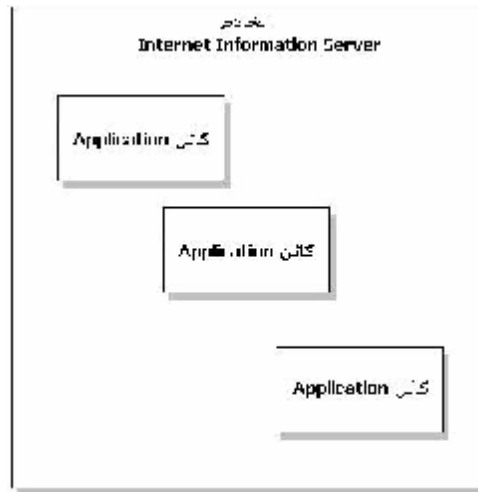
الكائن HttpServerUtility هو الخادم Server الذي يتم تنفيذ موقعك فيه، لا يحتوي هذا الكائن الا على خاصيتين الاولى MachineName تعود باسم جهاز الخادم، اما الخاصية الثانية ScriptTimeout فتستطيع ان تحدد فيها فترة Timeout بوحدة الثواني، وهذه الفترة تمثل أقصى مدة يمكن تنفيذ الصفحة عليها ومن ثم انتهاء تنفيذها ان تجاوزت الحد، تفيدك الطريقة السابقة عند الصفحات التي تقوم بتكوين حلقات لا نهائية -بطريق الخطأ- ولا تعود للمستخدم، او الصفحات التي تنجز مهام لجمل استعلام طويلة جدا مما تبطئ جهاز الخادم نفسه :

```
TextBox1.Text = Me.Server.MachineName
Me.Server.ScriptTimeout = 20
```

اما الطرق، فتوجد بها مجموعة من الطرق كالطريقة Execute() والتي يمكنك من تنفيذ صفحة aspx في داخل الصفحة الحالية.

الكائن HttpSessionState

انت تعلم ان الخادم IIS يمكنك من استضافة المواقع على جهازك، واذا نظرنا الى الموضوع بشكل تجاري، فان اغلب مواقع الاستضافة Hosting لا تقوم باستضافة كل موقع في جهاز خادم Server خاص، بل قد يحتوي الخادم على أكثر من موقع.
كيف يمكنك -كمبرمج ASP.NET- من التفريق بين موقعك والمواقع الأخرى على نفس الخادم IIS؟ والجواب هو عن طريق الكائن HttpSessionState الذي يمثل موقعك الخاص والمعطى لك من المستضيف (شكل 21-1).



شكل 21-1: الخادم IIS قد يحتوي على أكثر من تطبيق ASP.NET Application.

ملاحظة

ان كنت قد ثبت نسخة من الخادم IIS في جهازك الشخصي، فكل موقع في الدليل الوهمي الجذري http://localhost/sitename يمثل موقع ويحتوي على كائن HttpApplicationState مستقل.

يمكنك الاستفادة من هذا الكائن بتعريف قيم يتم مشاركتها بين جميع الزوار لموقعك:

```
Me.Application("المشرف المناوب") = "عباس السريع"
Label1.Text = Me.Application("المشرف المناوب")
```

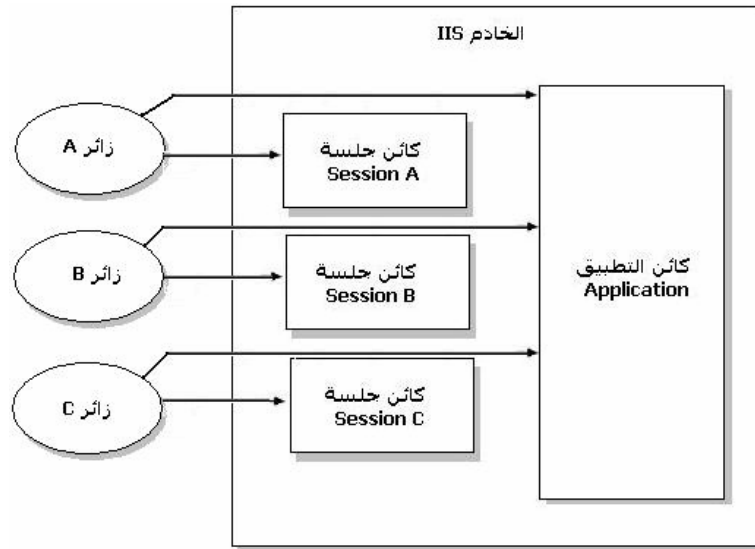
مع ذلك، عليك استدعاء الطرق Lock() و Unlock() قبل وبعد تعيين القيمة، وذلك حتى لا تسبب التعارضات عند استدعاء الصفحة من قبل أكثر من زائر بنفس الوقت:

```
Me.Application.Lock()
Me.Application("عدد الزوار") = Cint(Me.Application("عدد الزوار") + 1)
Me.Application.Unlock()
```

القيم التي أنشئت للتو تم حفظها في الخاصية الافتراضية Item() لذلك سمح لنا بتجاهلها، ولكن ضع في عين الاعتبار ان القيم السابقة ستحفظ من بداية تشغيل موقعك في المستضيف حتى نهايته وإيقافه من قبل المستضيف.

الكائن HttpSessionState

ان كان الكائن HttpSessionState يمثل موقعك، فان الكائن HttpApplicationState يمثل احد زوار موقعك، ويتم إنشاء كائن HttpSessionState لكل زائر من زوار موقعك بمجرد طلبه لأول صفحة من الصفحات (شكل 21-1).



شكل 21-2: كائن HttpSessionState لكل زائر.

التعامل مع الكائن HttpSessionState شبيه بالتعامل مع الكائن السابق HttpApplicationState، ولكن ضع في عين الاعتبار ان القيم التي تحفظ في الكائن HttpSessionState خاصة وتابعة لعمل واحد فقط، وكل عميل سيكون له نسخة خاصة من الكائن:

```
Me.Session("Name") = "عباس السريع"
Me.Session("Password") = "123"
```

توجد خاصية رائعة جدا وهي الخاصية IsNewSession التي تعود بالقيمة True ان كان الطلب للصفحة هو اول طلب للزائر، مما يعني كائن الجلسة HttpSessionState الخاص به جديد وتم إنشائه للتو:

```
If Me.Session.IsNewSession Then
    Me.Application("Counter") = Me.Application("Counter") + 1
End If
```

الملف Global.asax

عندما أنشأنا مشروع ASP.NET جديد في الفصل السابق، قامت بيئة التطوير Visual Studio NET. بإضافة الملف Global.asax ضمن قائمة الملفات في نافذة مستكشف الحل، وعليك معرفة ان كل تطبيق ASP.NET يحتوي -على الأكثر- ملف Global.asax واحد فقط يتم وضعه في المجلد الجذري للمشروع.

يقوم محرك ASP.NET في الخادم بتحميل الملف Global.asax عند بداية تنفيذ موقعك، وسيضل هذا الملف قيد العمل حتى يتم إيقاف عمل الموقع، وفي كل مرة تستدعى فيها صفحة من صفحات موقعك .aspx، يتم تنفيذ الشيفرة والموجودة في هذا الملف. تستطيع تعريف جميع المتغيرات او الثوابت العامة Global في الملف Global.asax لتتمكن من الوصول لها بين صفحات موقعك المختلفة، كما يمكنك الاستفادة من مجموعة من الإجراءات (تسمى أحداث أيضا) لها طابع خاص كما سترى في الفقرات التالية.

الإجراءات xxxStart() و xxxEnd()

عندما تفتح هذا الملف، ستجد ان بيئة التطوير قد عرفت الفئة Global تحتوي على مجموعة من الإجراءات:

```
Imports System.Web
Imports System.Web.SessionState

Public Class Global
    Inherits System.Web.HttpApplication

    Sub Application_Start(ByVal sender As Object, _
        ByVal e As EventArgs)

    End Sub

    ...
End Class
```

الإجراء Application_Start() يتم تنفيذه مع بداية تشغيل موقعك، وان تم إيقاف موقعك من قبل المستضيف سيتم تنفيذ الإجراء Application_End()، بينما الاجرائين Session_Start() و Session_End() سيتم تنفيذهما بمجرد انشاء كائن HttpSessionState جديد او إنهائه، اي - بعبارة اخرى- بمجرد دخول زائر جديد الى موقعك، لذلك يمكنك الاستفادة من هذه الإجراء - مثلاً- في معرفة عدد الموجودين حالياً في موقعك:

```
Public Class Global
    Inherits System.Web.HttpApplication

    ...
    ...
    Sub Application_Start(ByVal sender As Object, ByVal _
        e As EventArgs)

        Me.Application("CurrentVisitors") = 0
    End Sub

    Sub Session_Start(ByVal sender As Object, ByVal e As EventArgs)
        Me.Application.Lock()

        Me.Application("CurrentVisitors") = _
            CInt(Me.Application("CurrentVisitors") + 1)

        Me.Application.Unlock()
    End Sub

    Sub Session_End(ByVal sender As Object, ByVal e As EventArgs)
        Me.Application.Lock()

        Me.Application("CurrentVisitors") = _
            CInt(Me.Application("CurrentVisitors") - 1)

        Me.Application.Unlock()

    End Sub
End Class
```

الإجراء Global_Error()

عرفت في الفصل السابق كيف يمكن لصفحة النموذج Web Form من تعريف حدثها Error يتم تنفيذه بمجرد حدوث استثناء غير متفادى في الصفحة، مع ذلك لست بحاجة الى كتابة الشيفرة المطلوبة في كل ملف من ملفات صفحات موقعك -خاصة ان كان عددها كبيراً، اذ يكفي تعريفك

للإجراء Global_Error() والذي تكتب فيه الشيفرة المطلوبة ليتم تنفيذها بمجرد وقوع استثناء غير متفادى في أي صفحة من صفحات موقعك.

يمكنك معرفة الاستثناء الذي وقع عن طريق الخاصية GetLastError والتابعة للكائن Server، مع ذلك هذه الخاصية تعود دائماً بكائن استثناء HttpUnhandledException، لذلك ستحتاج إلى الوصول إلى الخاصية InnerException والتابعة لكائن الاستثناء:

```
Public Class Global
    Inherits System.Web.HttpApplication

    ...
    ...
    Sub Global_Error(ByVal sender As Object, ByVal e As EventArgs)
        Dim exp As Exception

        exp = Me.Server.GetLastError.InnerException

        Response.Write(exp.Message)
        Response.End()
    End Sub
End Class
```

الأمان Security

حسناً، موضوع الأمان موضوع كبير جداً، وعند ربطه بتطبيقات ASP.NET فإنه يشمل الخادم IIS ونظام التشغيل Windows، وبما أن هذا الكتاب لا يزال مختصاً ببرمجة Visual Basic .NET، فلن اتحدث بالتفصيل حول هذه المواضيع، إلا أنني ملزم بالتلميح إليها ولو باختصار - حتى تستوعب العلاقة بينها وبين صفحات ASP.NET.

مدخلك إلى الصلاحيات

حتى تفهم الصلاحيات عليك التفريق بين المصطلحين **التصديق Authentication** والسماح **Authorization**. بالنسبة للمصطلح الأول، فيمكننا تعريف التصديق على التعرف على العميل (الذي هو زائر الصفحة) والتصديق على هويته، وهذه بدو ذاتها تتعلق بحساب العميل User Account في خادم نظام التشغيل Windows Server، وأن لم يكن لهذا العميل حساب في نظام التشغيل، فيمكن أن يدخل كزائر Guest، وحينئذ نطلق على الطلبات بالطلبات المتاحة **Anonymous Requests**.

إذا اخذنا الموضوع من منظور شركات الاستضافة على الانترنت لمواقعك، فإن تنشئ هذه الشركات حسابات لكل زوار موقعك في خادماتها، لذلك فهي ستسمح بالطلبات المتاحة لتمكين الزوار من الدخول الى موقعك، ويمكن للخادم IIS من السماح لهذا النوع من الطلبات. بعد التعرف على العميل فان عملية التصديق Authentication تنتهي، وتأتي عملية أخرى وهي التفويض او السماح Authorization والتي تحدد فيها الصلاحيات للعميل الذي تم تعريفها للوصول الى المصادر، والمصادر لا يشترط ان تحصرها في صفحات aspx فقط، بل تشمل كافة مصادر جهاز الخادم كالملفات الموجودة في الخادم، او الوصول الى قواعد البيانات. خيارات التصديق والسماح يتم إعدادها من خلال اعدادات المستخدمين في نظام التشغيل Windows والخادم IIS، وفي اغلب الظن لن تكون لك قدرة في تغيير هذه الاعدادات الا ان كنت تعمل على جهازك الشخصي، او لك صلاحية المشرف Administrator على الشبكة المحلية، اما عند الحديث عن شركات الاستضافة لمواقع الانترنت، فلا اعتقد انك تستطيع تغيير شيء فيها. لذلك، ضع في عين اعتبارك دائما قضية الصلاحيات التي ستكون لك عند تصميم موقعك وتتوي استضافته على شبكة الانترنت من قبل شركات استضافة، فمعظم شيفراتك البرمجية لن تعمل كما هو متوقع بسبب تغيير الصلاحيات، تخيل مثلا ان احد صفحاتك تقوم بإنشاء ملف في القرص الصلب وأردت نقل ملفات الموقع على مستضيف لا يسمح لك بالقيام بهذه العملية.

أوضاع التصديق في ASP.NET

بعيدا عن الخادم IIS ونظام التشغيل Windows، دعنا نلقي الضوء على اساليب التصديق Authentication التي يمكنك السيطرة عليها والتحكم فيها لتعرض صفحات موقعك بالشكل المطلوب، والسبب الذي جعلني اذكرها في هذا الكتاب هو قدرتك على تحريرها والتحكم فيها من خلال ملفات التهيئة Configuration Files كما ستري لاحقا.

مشروع ASP.NET الذي تتجزه يمكن له ان يعتمد وضع او اسلوب من اربعة اوضاع هي None، Windows، Forms، و Passport. في الوضع الاول فانك لا تطلب من صفحات موقع اجراء اي عمليات تصديق Authentication او سماح Authorization، وستكتفي بالاعدادات الموفرة لموقعك من خلال الخادم IIS ونظام التشغيل Windows.

اما الوضع الثاني وهو Windows - فان عملية التصديق Authentication ستتم من قبل الخادم IIS ونظام التشغيل، ولكنك تستطيع التحكم في عملية السماح Authorization. يعيب هذه الطريقة، ان تعريف المستخدم (للتصديق عليهم) تتم باضافة حساباتهم من قبل نظام التشغيل Windows.

بالنسبة للوضع الثالث Forms، فيعطيك انت -كمطور الموقع- امكانية تحديد عملية التصديق Authentication والسماح Authorization بنفسك، وهو هذا هدفي من هذا القسم، لذلك خصصت فقرة كاملة ستراها قريبا.

اخيرا، في الوضع Passport فان عملية التصديق Authentication ستعتمد على خدمة Microsoft Passport، وهي خدمة عالمية تمكن المستخدمين من استخدام نفس الحساب User Account للدخول الى المواقع المختلفة، كخدمة Microsoft Messenger® تعتمد هذا على Microsoft Passport لتسجيل دخول الاعضاء (المزيد من التفاصيل حول هذه الخدمة، يمكنك زيارة الموقع <http://www.passport.com>).

ملفات التهيئة

قبل التحديث عن الوضع Forms للتصديق بودي تذكيرك بموضوع ملفات التهيئة، وتغيير طفيف يحدث فيه عند مشاريع ASP.NET.

في الفصل الحادي العشرات **المجمعات Assemblies** تحدثنا عن ملفات التهيئة Configuration Files، وذكرت ان لكل مجمع تنشئه يحتوي على ملف تهيئة التطبيق Application ينتهي بالامتداد config. تضع فيه كافة الخصائص التي تؤثر في سلوك تنفيذ المجمع.

اما عند الحديث عن تطبيقات ASP.NET، فان ملف تهيئة التطبيق يحمل الاسم web.config تضعه في المجلد الرئيسي للتطبيق، كما يمكنك تخصيص ملف تهيئة web.config لكل مجلد فرعي من مجلدات المشروع.

الوضع Forms للتصديق

من خصالي كمبرمج، فاني أفضل كثيرا وضع Forms للتصديق Authentication، ففيه احدد قائمة بالمستخدمين الذين اود التصديق عليهم للدخول الى الصفحات الحساسة والإدارية في موقعي، وذلك يتطلب مني -كمبرمج- من تصميم صفحة تسجيل الدخول Login Page، ليتمكن المستخدمين من كتابة اسماء معرفاتهم وكلمات المرور الخاصة بهم.

الوضع Forms للتصديق يعمل بالشكل التالي: عندما يقوم العميل بطلب صفحة aspx وكانت هذه الصفحة مخصصة للمصدق عليهم Authenticated فقط، سيقوم محرك ASP.NET بالتحقق من تذكرة التصديق Authentication Ticket -والتي قد تكون محفوظة في الـ

Cookies مثلا، ان تم التحقق من تذكرة التصديق فسيتم عرض الصفحة المطلوبة، واذا لا فسيتم تحويل الزائر الى صفحة تسجيل الدخول.

ان اردت تطبيق التصديق Authentication في موقعك بالصورة التي رسمتها لك، اول خطوة عليك تنفيذها هي فتح ملف التهيئة web.config والبحث عن الوسم authentication والوسم authorization:

```
<configuration>
  <system.web>
    ...
    ...
    <authentication mode="Windows" />

    <authorization>
      <allow users="*" />
    </authorization>
    ...
    ...
  </system.web>
</configuration>
```

قم بتغيير وضع التصديق من Windows الى Forms، اكتب بداخله الوسم <forms> واضف به هذه المواصفات:

```
<configuration>
  <system.web>
    ...
    ...
    <authentication mode="Forms">
      <forms loginUrl="/login.aspx" name="dev4arabs" timeout="10"/>
    </authentication>
    ...
    ...
  </system.web>
```

اهم مواصفة في الوسم <forms> السابق، هي المواصفة loginUrl والتي تحدد فيها الصفحة التي سيتم توجيه المستخدم لها ان لم يتم التصديق عليها، بالنسبة المواصفات الاخرى فمعظمها لها قيم افتراضية وتقل أهميتها ولكني انصحك دائما باستخدامها، فالمواصفة name تحدد فيها اسم الـ Cookie الذي سيتم اعتباره كتذكرة تصديق وسيتم التحقق منها عند كل طلب لصفحات موقعك، اما المواصفة timeout فهي تفيدك لتحديد الفترة -بالدقائق- الغير نشطة ليتم انتهاء جلسة كائن العمل Session وكأنه سجل خروجه.

اخيرا، جميع الزوار يمكنهم الدخول الى صفحات موقعك، وذلك بسبب الوسم الفرعي `<allow>` والتابع للوسم `<authorization>`، اذ عليك كتابة الوسم `<deny>` حتى تمنع الطلبات المتاحه Anonymous Request:

```
<configuration>
  <system.web>
    ...
    ...
    <authentication mode="Forms">
      ...
    </authentication>

    <authorization>
      <deny users="?" />
    </authorization>

    ...
  </system.web>
```

الوسم <credentials>

في مشاريعك الجدية، فان اسماء الاشخاص وكلمات مرورهم سيتم قراءتها وحفظها في ملف قاعدة بيانات، وعليك كتابة كافة الشيفرات الضرورية في صفحة تسجيل الدخول للتحقق من اسم المستخدم وكلمة المرور ومن ثم تسجيل دخوله، اما هنا فسنستخدم الوسم `<credentials>` لتسهيل العملية وكتابة اسماء الاعضاء وكلمات مرورهم في داخل الوسم الفرعي `<forms>` التابع للوسم `<authentication>`:

```
<configuration>
  <system.web>
    ...
    ...
    <authentication mode="Forms">
      <forms ... ..>
        <credentials passwordFormat="Clear">
          <user name="عباس السريع" password="123" />
          <user name="زكريا زعتي" password="456" />
        </credentials>
      </forms>
    </authentication>
    ...
  </system.web>
```

بالنسبة للمواصفة passwordFormat ففيها تحدد خوارزم التشفير، وبما أننا لا نود استخدام اي خوارزم هنا، فوضعنا القيمة "Clear" لها.

صفحة تسجيل الدخول Login

والآن عليك تصمم صفحة تسجيل الدخول وتحفظها بنفس الاسم الذي حددته في الخاصية loginUrl في الوسم <forms> السابق، قم بتصميم صفحة بإضافة أداتي TextBox لكتابة اسم المستخدم وكلمة المرور (شكل 21-3) - لا اعتقد أنك بحاجة الى درس في تصميم صفحات تسجيل الدخول!



شكل 21-3: صفحة تسجيل دخول.

كما ذكرت سابقاً، في مشاريعك الجديدة ستعتمد على قاعدة بيانات لقراءة اسم المستخدم وكلمة المرور، وبما أننا اعتمدنا على الوسم <credentials> في الفقرات السابقة، فسنستدعي الطريقة المشتركة Authenticate() والتابع لفئة System.Web.Security.FormsAuthentication. تتطلب هذه الطريقة اسم المستخدم وكلمة المرور، وستعود بالقيمة True ان تم التحقق منها:



```
Imports System.Web.Security
...
...
Private Sub Button1_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles Button1.Click

    If FormsAuthentication.Authenticate(txtName.Text, _
        txtPassword.Text) Then

        FormsAuthentication.RedirectFromLoginPage(txtName.Text, True)
    Else
        Label1.Text = "خطأ في كلمة المرور"
    End If
End Sub
```

عند طلب اي صفحة من صفحات موقعك، سيتم توجيه الزائر الى صفحة تسجيل الدخول، وان تم التصديق عليه، سيتم نقله الى الصفحة التي طلبها في اول مرة- باستخدام الطريقة `RedirectFromLoginPage()` والتي تسند لها القيمة `True` في وسطها الثانية ليتم حفظ الـ Cookie في جهاز المستخدم.

مواضيع متقدمة

سأتحدث في هذا القسم عن مجموعة متفرقة من المواضيع: كالتخزين المؤقت `Caching`، التصريح عن المتغيرات العامة، حماية الصور وعرضها ديناميكيا، ووحدات `Http Modules`.

التخزين المؤقت `Caching`

القلب النابض لصفحات موقعك المبنية على تقنية ASP.NET ستقوم بتوليد صفحات HTML بعد قراءة بياناتها من قواعد بيانات، مع ذلك فان اغلب صفحات موقعك لن يتم تحديثها بشكل دوري الا كل فترة طويلة، لذلك يفضل تخزين **Caching** صفحات موقعك التي لا تتوقع تحديثها في اجهزة الزوار، حتى يخفف الضغط على جهاز الخادم `Server`.

يمكنك تخزين الصفحة في جهاز العميل باستخدام الموجه `@OutputCache` والذي تكتبه في اعلى ملف `aspx` وترفق معه الفترة -بالثواني- التي تود من تخزين الصفحة في الجهاز، وبعدها سيتم تحديث `Refresh` الصفحة عندما يزورها المستخدم مرة اخرى -ولا تنسى استخدام الموصوفة `VaryByParam`:

```
<%@ OutputCache Duration="10" VaryByParam="none"%>
```

بعد اضافتك للسطر السابق للصفحة، سيتم تخزين الصفحة في جهاز العميل ولن يتم تحديثها الا كل 10 ثواني، وللتحقق من ذلك اضع أداتي Label و Button واكتب شيئاً مثل:

```
Private Sub Button1_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles Button1.Click

    Label1.Text &= Date.Now & "<br>"
End Sub
```

عند تنفيذ الصفحة، ستلاحظ انك في كل مرة تضغط فيها على الزر لن يتم تحديث الصفحة الا كل عشرة ثواني (شكل 21-4 أ بالصفحة التالية)، اما إن ألغيت الموجهه @OutputCache فسيتم تحديث الصفحة في كل مرة يطلبها الزائر (شكل 21-4 ب). بالنسبة للمواصفة VaryByParam ففيها تمكن عملية التخزين المختلفة لنفس الصفحة بحسب القيم المرسله لها، فمثلا راقب الموجه التالي:

```
<%@ OutputCache Duration="10" VaryByParam="id"%>
```

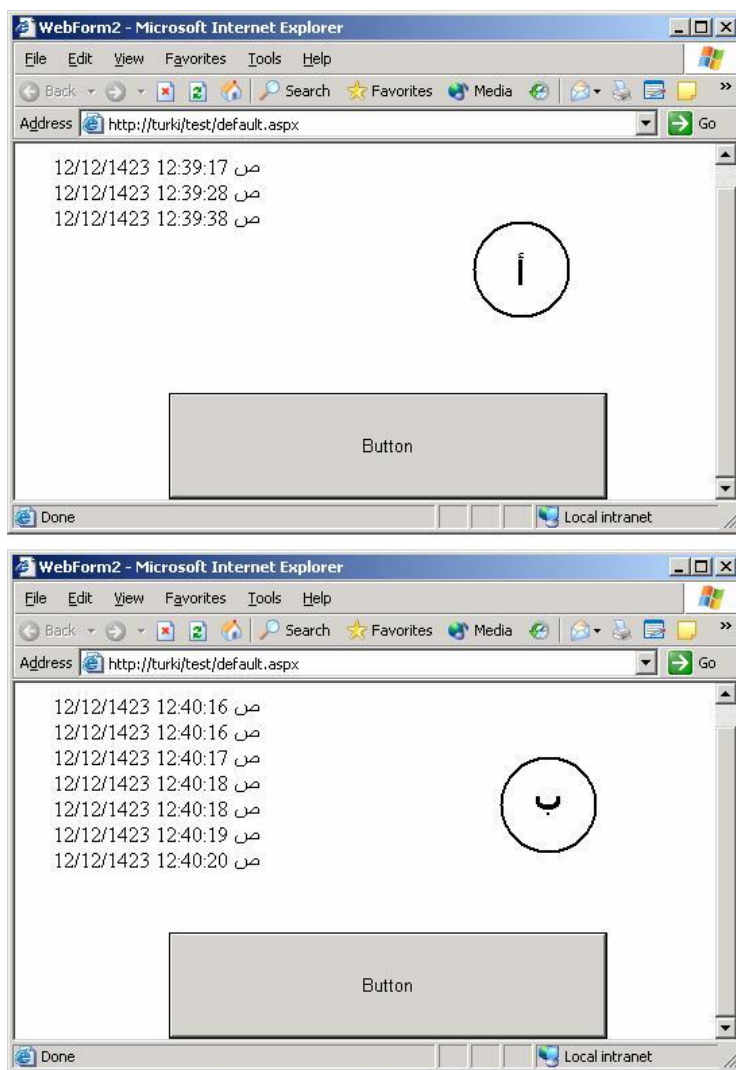
هـب مثلا ان الزائر قد ادخل الروابط التالية للصفحة السابقة:

```
http://www.dev4arabs.com/testcache.aspx?id=1
http://www.dev4arabs.com/testcache.aspx?id=1
http://www.dev4arabs.com/testcache.aspx?id=1
http://www.dev4arabs.com/testcache.aspx?id=2
http://www.dev4arabs.com/testcache.aspx?id=2
http://www.dev4arabs.com/testcache.aspx?id=3
```

استنادا الى الروابط السابقة، فسيتم تخزين ثلاث نسخ من الصفحة، باختلاف القيم التي تحملها روابطها، ولن يتم تحديثها الا بعد مرور 10 ثواني او كتابة قيمة جديدة في القيمة المرسله id بالرايط.

ملاحظة

الطلبات السابقة أرسلت الصفحة بأسلوب الـ GET، مع ذلك يمكنك تحديد القيم المرسله بأسلوب الـ POST.



شكل 21-4: الصفحات المخزنة Cached لا يتم تحديثها الا بعد فترة.

أخيرا، يمكنك تحديد مكان تخزين الصفحة، اما في جهاز العميل Client، الخادم Server، او في جهاز آخر - غير الخادم Server - أدى الى توصيل او استقبال الطلب Downstream (يكون في اغلب الاحوال خادم البروكسي Proxy). تستطيع تحديد المكان عن طريق المواصفة Location، وان تجاهلتها فستكون قيمتها Any:

```
<%@ OutputCache ... Location="Downstream"%>
<%@ OutputCache ... Location="Server"%>
<%@ OutputCache ... Location="Client"%>
<%@ OutputCache ... Location="Any"%>
```

المتغيرات العامة

سابقا في هذا الفصل، تعلمنا استخدام الكائن Application للاحتفاظ بالقيم التي نود الوصول لها من الصفحات المختلفة للموقع، مع ذلك انصحك بشدة بالاعتماد على المتغيرات حيث ان الوصول لها اسرع بمئات المرات، ولن تحتاج الى استخدام دوال التحويل معها، فبدلا من كتابة شيئا مثل:

```
Application("Counter") = CInt(Application("Counter") + 1)
```

يمكنك تعريف وحدة برمجية Module في اي ملف من ملفات موقعك وتعريف متغير بها:

```
Module MainModule
    Public Counter As Integer
End Module
```

وسيتم الوصول لها بالشكل التقليدي من اي صفحة في صفحات موقعك:

```
Counter += 1
```

حماية الصور

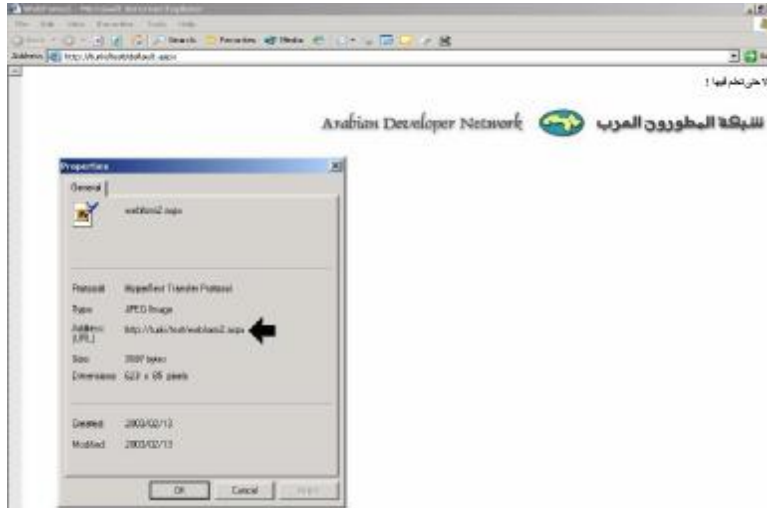
قد تحمل ملفات موقعك باحد شركات الاستضافة، وكانت هذه الشركة تحددك على مقدار تدفق بيانات Bandwidth محدد، واكتشفت لاحقا بان بعض الأشخاص -الغير لبقين - كانوا يستخدمون روابط الصور في موقعك وعرضها في مواقعهم، مما كلفك الكثير من مقدار تدفق البيانات - ومقدار تدفق أموال محفظتك ايضا!

لا دموع بعد اليوم، فالكائن Response يحتوي على الخاصية ContentType والتي يمكنك من تغيير هيئة الصفحة، فصفحات aspx ترسل صفحات HTML بصيغة نصية بشكل ابتدائي، ولكنك تستطيع تحويله الى هيئة صور من نوع JPG:



```
Dim bmp As Bitmap = Bitmap.FromFile("dev4arabs.jpg")
Dim showPic As Boolean = True

' اكتب أي شرط هنا لعرض الصورة
If showPic Then
    Response.ContentType = "image/jpeg"
    bmp.Save(Me.Response.OutputStream, Imaging.ImageFormat.Jpeg)
End If
```



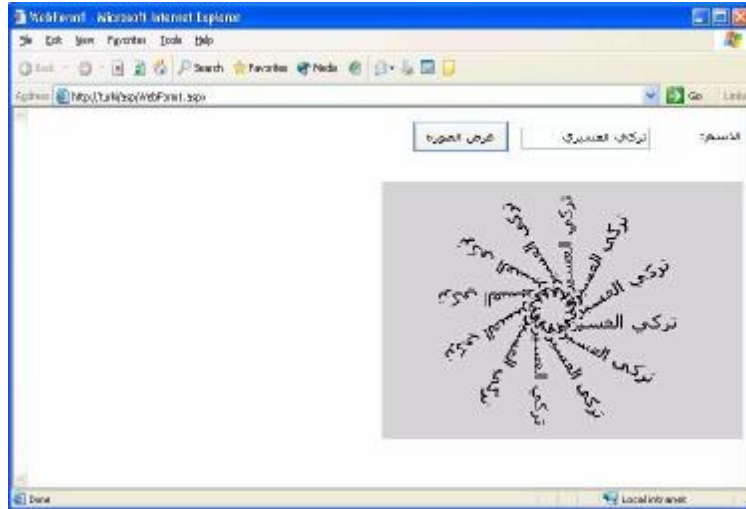
شكل 21-5: لن يتمكن احد من استعمال رابط الصورة إلا ان تحقق الشرط.

لاحظ ان رابط الصورة في (الشكل 21-5) يشير الى صفحة aspx وليس ملف صورة JPG. وبالنسبة لملف الصورة عليك وضعه في مجلد لا يمكن أن يصل اليه الزائر بكتابة رابط URL والخاص به.

التوليد الديناميكي للصور

مهلاً مهلاً! إلا تلاحظ أننا استخدمنا تقنية GDI+ في الفقرة السابقة؟ إذا كان هذا لا يعني لك

شيئاً فما رأيك (بالشكل 21-6):



شكل 21-6: التوليد الديناميكي للصور.

الفكرة من الصفحة السابقة هي عملية توليد الصور بشكل ديناميكي - وقت التنفيذ - وذلك باستخدام

فئات GDI+:



```
Dim bmp As New Bitmap(400, 300,
Drawing.Imaging.PixelFormat.Format16bppRgb565)
Dim gr As Graphics = Graphics.FromImage(bmp)
Dim Font As New Font("Tahoma", 16)
Dim text As String = "تركيب الحرفي"
gr.Clear(Color.LightGray)

' عملية القلب '
Dim angle As Single
For angle = 0 To 360 Step 30
    gr.ResetTransform()
    gr.TranslateTransform(200, 150)
    gr.RotateTransform(angle)
    gr.DrawString(text, Font, Brushes.Black, 0, 0)
Next
```

```
Response.ContentType = "image/jpeg"
bmp.Save(Response.OutputStream, Imaging.ImageFormat.Jpeg)

' قتل الكائنات
Font.Dispose()
gr.Dispose()
bmp.Dispose()
```

بعدما عرفتكم على أساسيات تطوير صفحات ASP.NET، لم يتبقى لنا الا عرض احد المزايا الجديدة في إطار عمل .NET Framework. وهي **خدمات ويب Web Services** عنوان الفصل التالي.

خدمات ويب Web Services

اختم معك الكتاب بهذا الفصل والذي يتحدث عن خدمات ويب وطريقة بنائها، ولا تستغرب من قلة عدد الصفحات المخصصة لهذا الفصل، فخدمات ويب ما هي إلا تطبيقات ASP.NET تقليدية، وكل ما تعلمناه في الفصول السابقة يمكنك تطبيقه مع خدمات ويب.

ملاحظة

المشاريع من نوع Web Services تدرج مجموعة من مجالات الأسماء التابعة لها، وإن لم تحدد هذا النوع من المشاريع، فقد تحتاج الى استيراد مجال الاسماء التالي:

```
Imports System.Web.Services
```

مدخلك إلى خدمات ويب

كما ذكرت في الفصل الأول تعرف على **Visual Basic .NET**، خدمة ويب (تسمى أحياناً **XML Web Service**) ما هي إلا برنامج يستقبل طلبات **Requests** ومن ثم تستجيب لها **Response** باستخدام بروتوكول **HTTP** تحت معايير لغة **XML** القياسية، وبذلك تتمكن ملايين المواقع المنتشرة حول العالم من تبادل البيانات فيما بينها وإنجاز الأعمال المطلوبة. كمبرمج **Visual Basic .NET**، لست بحاجة إلى أي خبرة عملية في بروتوكول الاتصال **HTTP** أو لغة الاستعلام **XML**، فالمشاريع من النوع **Web Services** تنجزها بنفس الشيفرة البرمجية بلغة **Visual Basic .NET**.

كيف تعمل خدمة ويب؟

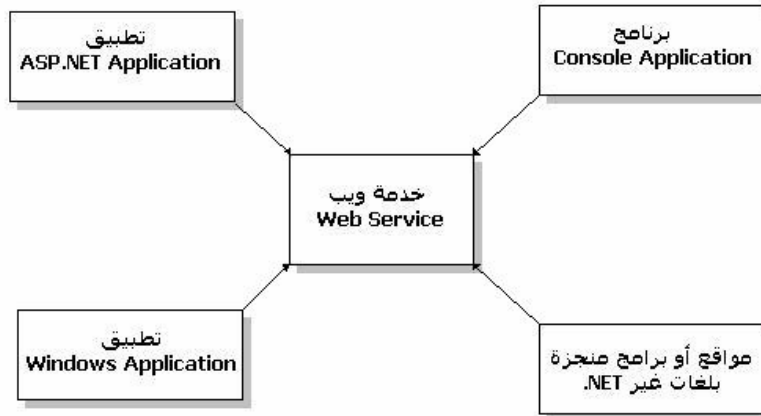
حتى أسهل عليك فكرة خدمات ويب، يمكنك اعتبار خدمة ويب على انها مكتبة DLL تستخدم فئاتها لتنشئ كائنات منها في برامجك، ولكن هذه المكتبة ليس في جهازك الشخصي و إنما على موقع تصل اليه بشبكة الانترنت.

يمكنني -مثلا- تطوير فئة باسم VBNETBook تحتوي على طريقة GetUpdates() تعود هذه الطريقة بمصفوفة حرفية تمثل آخر الاخطاء والتحديثات المتعلقة بهذا الكتاب، لتتمكن من استدعائها بـ Visual Basic .NET بنفس الطرق التقليدية:

```
Dim book As New VBNETBook
Dim x As String

For Each x In VBNETBook.GetUpdates()
    ArabicConsole.WriteLine ( x )
Next
```

نستنتج من ذلك، ان خدمات ويب هي الوسيلة المثلى التي تمكن تطبيقات ومواقع .NET من الحصول وتبادل البيانات، دون الحاجة الى أي خبرة مسبقة في بروتوكول الاتصال HTTP ولغة وصف البيانات XML (شكل 22-1):



شكل 22-1: يمكن استخدام خدمة ويب من أي برنامج أو موقع آخر.

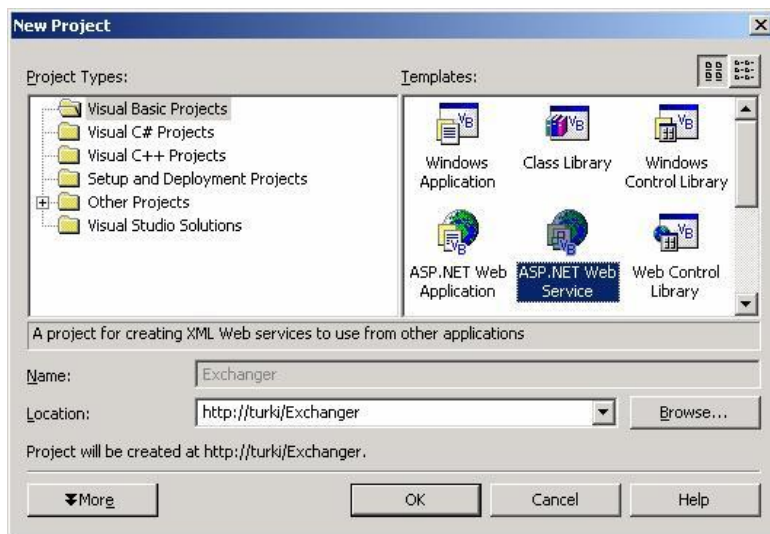
كما تلاحظ في الشكل السابق، يمكن أيضا للمواقع والتطبيقات المنجزة بلغات غير .NET من استخدام خدمات ويب، فهي ليست سوى موقع يرسل ويستقبل الطلبات ببروتوكول الاتصال HTTP ولغة وصف البيانات XML.

بناء خدمة ويب

يكفينا من الكلام حول خدمات ويب وما الذي يمكنه ان تقدمه لك، سنقوم في هذا القسم ببناء اول خدمة ويب والتي تقدم لك خدمة تحويل العملات، لترسل لها مبلغ لعملةك المحلية، وستعود بالمقابل لها بالريال السعودي.

إنشاء المشروع

كما ذكرت سابقا، لا تختلف الفكرة بين خدمات ويب عن المشاريع ASP.NET، الا ان بعض الملفات والتابعة لمشاريع ASP.NET لن تحتاجها، ولانشاء خدمة ويب جديدة اختر الامر New-Project > من قائمة File، وحدد الرمز ASP.NET Web Service من القوائم، ثم ضع اسم موقعا مناسب لمشروعنا كـ <http://localhost/Exchanger> (شكل 2-22).

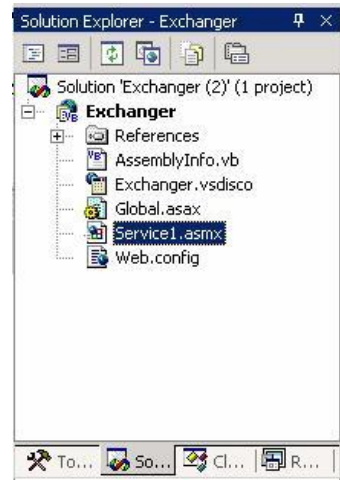


شكل 2-22: تحديد مشروع لخدمة ويب ASP.NET Web Service.

ملاحظة

لا تنسى تثبيت الخادم Internet Information Server، ان كنت تنوي تجربة خدمة ويب على جهازك الشخصي.

اضغط على الزر OK، ستقوم بيئة التطوير Visual Studio .NET ببناء ملفات مشروعك مشابهه لملفات مشاريع ASP.NET، فيوجد بها الملف web.config والملف global.asax (شكل 22-3) والذان يعملنا بنفس الطريقة في مشاريع ASP.NET التقليدية.



شكل 22-3: ملفات مشروع خدمة ويب.

بالنسبة للملف Service1.asmx فهو الملف الرئيسي للخدمة التي ستكتب بها الشفرات، ويمكنك اعتبارها كمكتبة فئات Class Library يتم الاتصال بها عن طريق الانترنت باستخدام بروتوكول HTTP.

غير اسم الملف من Service1.asmx الى اسم مناسب، والسبب ان اسم الملف سيتم استخدامه برمجيا ويمكننا الوصول له لاحقا، ضع الاسم SaudiRiyals.asmx مثلا.

كما هو الحال مع ملفات aspx، الملف asmx تابع لمصمم يمكنك من اضافة وإدراج أدوات عليه من صندوق الأدوات ToolBox، مع ذلك فإن خدمة ويب لا يفترض ان تستخدم أدوات مبرمجة وذلك لأنها لا تعرض واجهة استخدام. يمكنك وضع أداة لا تعرض كالأداة FileSystemWatcher (وهي نسخة على شكل أداة من الفئة System.IO. FileSystemWatcher التي استخدمناها في الفصل السادس عشر **مواضيع متقدمة** لإنشاء خدمة (Windows).

كتابة الشيفرة

الملف SaudiRiyals.asmx يحتوي على فئة بالاسم Service1، وهي تمثل الخدمة التي تصممها، أي بعبارة أخرى- أي فئة مشتقة وراثياً من الفئة System.Web.Services.WebService تمثل خدمة ويب. غير اسم الفئة من الخاصية (Name) في نافذة الخصائص إلى SaudiRiyals، ثم انقر نقرا مزدوجا على مصمم الخدمة لفتح نافذة محرر الشيفرة، وقم بتعريف الاجرائين FromEG() و FromBRN():



```
Imports System.Web.Services

<System.Web.Services.WebService(Namespace:="http://tempuri.org/") > _
Public Class SaudiRiyals
    Inherits System.Web.Services.WebService

    #Region " Web Services Designer Generated Code "
    ...
    ...
    #End Region

    <WebMethod(Description:="تحويل المبلغ من الجنيه المصري الى الريال السعودي")>
    Function FromEG(ByVal amount As Decimal) As Decimal
        Return amount * 0.9
    End Function

    <WebMethod(Description:="تحويل المبلغ من الدينار البحريني الى الريال السعودي") >
    Function FromBRN(ByVal amount As Decimal) As Decimal
        Return amount * 10
    End Function

End Class
```

الاجرائين FromEG() و FromBRN() هما الاجرائين الرئيسيين للخدمة، الاول يقوم بتحويل المبلغ المرسل بالجنيه المصري الى الريال السعودي، والثاني من الدينار البحريني، وكلا القيم من النوع Decimal.

ملاحظة

انا مبرمج ولست مصرفيا، ولم اعمل في القطاعات المالية ابدا في حياتي، لذلك تجاهل النسب الخاطئة والمستخدمة في عملية الضرب لتحويل المبلغ فالغرض هو التوضيح فقط.

بالنسبة للموصفة WebService المسطورة في اعلى الفئة والموصفة WebMethod في اعلى الاجرائين، فاعرف-بشكل مبسط- ان الاولى تجعل الفئة فئة خدمة ويب Web Service Class، والثاني تجعل الإجراء قابل للوصول من خلال شبكة الانترنت.

اختبار الخدمة من المتصفح

بمجرد كتابة الشيفرة السابقة وتعريف الاجرائين، فان الخدمة SaudiRiyals أصبحت جاهزة، ويمكنك اختبارها عن طريق المتصفح Internet Explorer® وذلك بتنفيذ البرنامج والضغط على المفتاح [F5] (شكل 22-4).



شكل 22-4: اختبار الخدمة من المتصفح® Internet Explorer.

تلاحظ في أعلى شريط العنوان بالمتصفح، بان رابط URL يشير الى ملف الخدمة SaudiRiyals.asmx، والذي تم تعريف الفئة SaudiRiyals فيه، كما يظهر لك في اعلاه اسم الخدمة والاجرائين FromEG() و FromBRN() اللذان عرفناهما للتو، بالإضافة الى نفس الوصف الذي كتبناه في المواصفه WebMethod لكلا الاجرائين.

كيف تم الحصول على هذه المعلومات؟ والجواب قام محرك ASP.NET باستخدام فئات الانعكاس Reflection Classes لعرض الطرق التي تحتويها الخدمة، كما تفعل تماماً بيئة التطوير Visual Studio .NET. بعد كتابتك لنقطة بعد اسم الكائن ليتم عرض جميع طرقه وخصائصه.

ملاحظة

ان كان الملف يحتوي على اكثر من فئة لخدمة ويب، سيتم عرض الفئة الاولى فقط في المتصفح Internet Explorer.

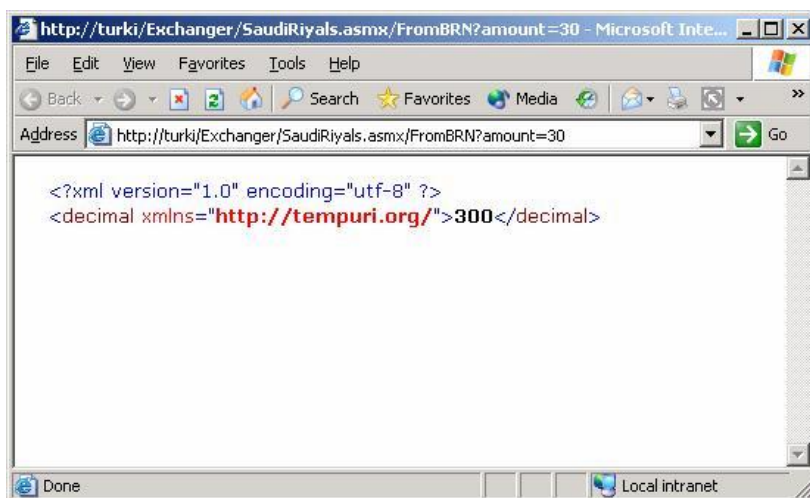
تطبيقاً، لن يتم الوصول الى هذه الخدمة بهذه الطريقة، اذ ان المبرمجين سيصلون الى طريقها من داخل شيفراتهم المصدريّة، ولكن الميزة التي يقدمها إطار عمل .NET Framework في عرض الخدمة على المتصفح ستفيدك كثيراً، وستسهل عليك تجربة واختبار الخدمة قبل اعتمادها ونشرها في كل ارجاء الارض عن طريق شبكة الانترنت.

دعنا الان نقوم بتجربة لاستدعاء احد الاجراءات، اضغط على الرابط FromBRN ليتم نقلك الى صفحة يمكنك من اسناد قيم لوسيطات الإجراء (شكل 22-5).



شكل 22-5: كتابة وسيطات الإجراء (FromBRN).

ضع في عين الاعتبار، ان ليس كل القيم يمكنك اسنادها من خلال المتصفح لتجربة الخدمة، فمثلاً قيم الكائنات Object والمعرفة من الفئات Classes على وجه التحديد لن تستطيع تجربتها وارسالها الى الاجراء كما في الشكل السابق، ولكن معظم انواع البيانات الابتدائية Primitive Types (ك Integer، Long، String، Decimal، ... الخ) يمكنك تجربتها، ادخل القيمة 30 للوسيطه amount واضغط على الزر Invoke لتظهر لك القيمة التي يعود بها الاجراء (شكل 22-6 بالصفحة المقابلة).



شكل 22-6: عاد الإجراء بالقيمة 300 بعد تحويل العملة للمبلغ المرسل.

من الشكل السابق يتضح لنا ان الخدمة تعمل بنجاح ويمكنك اعتمادها، جرب اختبار الاجراء الاخر FromEG() وتأكد بنفسك نتيجة عملية التحويل، وستعود القيمة بالصيغة XML، لانه كما ذكرت لك في بداية هذا الفصل - ان الغرض الرئيسي من خدمات ويب هو تبادل البيانات بين المواقع المختلفة بصيغة XML.

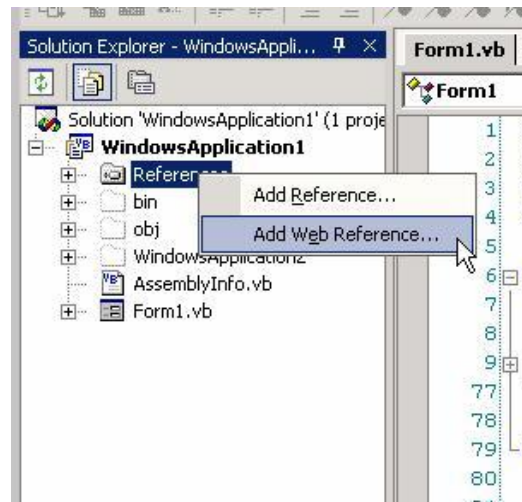
استخدام الخدمة

لو عدنا الى تعريف خدمة ويب في مقدمة هذا الفصل، ستذكر ان خدمة ويب ما هي الا برنامج يستقبل الطلبات Requests ويعود بالاستجابات Responses بصيغة XML تحت بروتوكول HTTP، وهذا يعني اي برنامج او عملية يستخدم البروتوكول HTTP يمكن له استخدام الخدمة، مما يعني انه لا يشترط ان يكون العميل يستخدم برنامج منجز باحد لغات NET، ليس هذا فقط بل لا يشترط ايضا ان يكون تطبيق تحت بيئة Windows، فلو كنت من مبرجي Macintosh، Linux، او حتى Unix وكنت على دراية كافية ببروتوكول الاتصال HTTP ولغة XML، يمكنك استخدام الخدمة دون اي مشاكل.

اما في هذا الكتاب سأريك كيف تستخدم الخدمة من برنامجك المكتوب بلغة Visual Basic .NET، ويمكنك استخدام الخدمة من اي نوع من المشاريع كـ Windows Service،

ASP.NET Application، Class Library، وغيرها وفي الخطوات التالية سنجرب الخدمة من مشروع Windows Application.

أنشئ مشروع من النوع Windows Application، وكما تفعل مع مكتبات الفئات التي تدرجها من نافذة المراجع، سنقوم بإدراج مرجع لخدمة ويب التي صممناها للتو، وحتى يمكنك إضافة مرجع لهذه الخدمة، انقر بزر الفأرة الأيمن على رمز المشروع في نافذة مستكشف الحل Solution Explorer واختار هذه المرة الأمر Add Web Reference - وليس Add Reference (شكل 22-7).



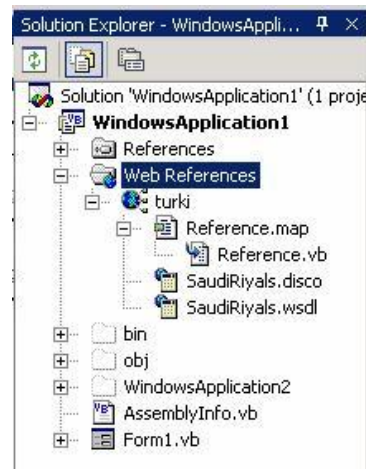
شكل 22-7: إضافة مرجع لخدمة ويب.

بمجرد اختيارك للامر السابق، سيظهر لك صندوق حوار Add Web Reference يمكنك من البحث عن الخدمات في شبكة الانترنت، في اعلى صندوق الحوار اكتب المسار الرابط الكامل للملف asmx الذي يحتوي على الخدمة (سـيكون <http://localhost/Exchanger/SaudiRiyals.asmx> ان اتبعت الخطوات السابقة) ثم اضغط على المفتاح [Enter]، وبعدما يتم الاتصال بالخدمة ستعرض لك أعضائها في جهة اليسار من صندوق الحوار (شكل 22-8).



شكل 22-8: صندوق حوار Add Web Reference.

ان تم اصطياذ الخدمة بالشكل الصحيح، فاضغط على الزر Add Reference في اسفل صندوق الحوار، سيتم الاتصال بالخدمة ومن ثم اضافتها الى خانة مراجع الخدمات Web References في نافذة مستكشف الحل Solution Explorer (شكل 22-9).



شكل 22-9: تم إضافة مرجع لخدمة ويب SaudiRiyals.

ملاحظة

لن تظهر لك كافة الملفات (بالشكل 22-9) الا ان حددت الاختيار Show All files في اعلى نافذة مستكشف الحل Solution Explorer.

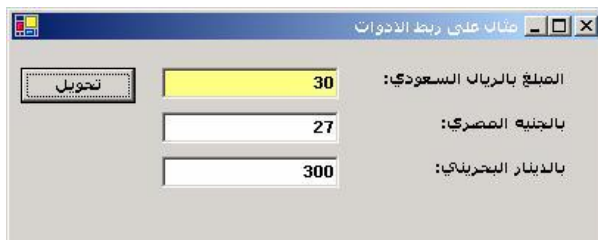
والان تم اضافة مرجع الى هذه الخدمة، وان كنت لا تصدق ما تراه قم باضافة ادوات TextBox و زر Button، واكتب هذه الشيفرة في الحدث Click للزر:

```
Private Sub Button1_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles Button1.Click

    Dim test As New LocalHost.SaudiRiyals()

    txtEG.Text = CDec(test.FromEG(txtSaudi.Text))
    txtBRN.Text = CDec(test.FromBRN(txtSaudi.Text))
End Sub
```

جرب تنفيذ البرنامج، وقم بإدخال المبلغ بالريال السعودي في أداة TextBox المناسبة، ومن ثم اضغط على الزر Button لترى نتائج تحويل المبلغ الى العملات الأخرى، وهذا يعني عملية الاتصال بالخدمة تمت بنجاح واستخدمت بالشكل المطلوب (شكل 22-10).



شكل 22-10: استخدام خدمة ويب في مشروع Windows Application.

تحديث الخدمة

عندما تجري أي تعديل على خدمة ويب -خاصة في مرحلة التجربة- عليك تحديثها في برنامج العميل دائماً (وهو تطبيق Windows Application كما في المثال السابق) حتى تظهر التعديلات ويتم تنفيذ الخدمة بالشكل الصحيح. لعمل ذلك، انقر بزر الفأرة الأيمن على رمز الخدمة في نافذة مستكشف الحل (شكل 22-9 بالصفحة السابقة)، واختار الأمر Update Web Reference من القائمة المنبثقة.

الصفحة 735 هي نهاية الفصل الثاني والعشرون **خدمات ويب Web Services**، وفي نفس الوقت نهاية الجزء الخامس **برمجة ويب**، والتي تمثل نهاية كتاب **برمجة إطار عمل .NET**. باستخدام **Visual Basic .NET**. أسأل الله العظيم بأسمائه الحسنى وصفاته العلى أن يجعله من العلم الذي ينتفع به، انه سميع مجيب الدعوات.

لغة وصف البيانات XML

انتشرت لغة وصف البيانات eXtensible Markup Language انتشارا لا مثيل له بين كافة التطبيقات التجارية على اوسع نطاقها، ولو تمتعت قليلا في إطار عمل .NET Framework. لاكتشفت ان البنية التحتية لاغلب ادواتها وفئاتها تعتمد XML كلغة قياسية لوصف البيانات. يمكنك عرض مواقع الانترنت المختلفة للحصول على مراجع وصيغ استخدامات لغة وصف البيانات XML، ولكنني في هذا الملحق اعرض لك مقرر سريع للغة وصف البيانات XML، أخطب به أولئك المبرمجين الذي لم يتشرفوا بمعرفتها لا من قريب ولا من بعيد.

استخدام وسوم HTML

ان كنت من مصممي مواقع ويب المعتمدة على صفحات HTML فقد تعتقد ان استخدام وسوم HTML Tags كافي لتبادل البيانات بين المواقع، ولكن الحقيقة غير ذلك، والسبب ان وسوم HTML غرضها تنسيق Formating البيانات على المتصفحات، وهي لا تمثل البيانات بشكلها الاستقلالي.

فمثلا، لو وجدت وسوم HTML التالية في احد الصفحات:

```
<p>تركبي العسيري</p>
<p>99</p>
```

ماذا تفهم من هذه البيانات؟ لاشئ والسبب قد يبدو منطقيا ان علمت ان وسوم HTML لا تشرح ولا توضح البيانات وانما تقوم بعرضها فقط، فقد يقوم احد الاشخاص باضا فت الوسوم السابقة في موقعه بهذا الشكل:

```
<p>تركبي العسيري</p> <p>الميرمج<b></b></p>
<p>99</p> <p>العمر<b></b></p>
```

وقد يقوم اخر بتفسيرها بهذا الشكل:

<p>الطباخ</p> <p>تركبي العسيري</p>
<p>الوزن</p> <p>99</p>

كيف ستقوم بتبادل البيانات مع التطبيقات والمواقع المختلفة ان لم تكن انت اصلا لديك اي خلفية عن الماهية الحقيقية للبيانات وتركيباتها الشكلية، وقد يصل الامر ايضا لحصولك على المعلومات الغير صحيحة، فمثلا ان تبادل بياناتك مع موقع يحتوي على الوسوم السابقة، قد تحصل على معلومات لا تعني لك شيئا:

تركبي العسيري : الاسم </p> غ

ويا ليت المشكلة تقف عند النقطة السابقة، فلو قرأت صفحة تحتوي على وسوم HTML وطلبتك تجريد نصوصها ومعلومات، سيكون الامر مقدور عليه -في قليل من الأحيان، فمثلا الوسم التالي:

<p>تركبي العسيري</p>

يمكنك تجريده بحذف الثلاث الحروف الأولى والأربع الأخيرة من الوسم، لتصبح النتيجة:

تركبي العسيري

تخيل مثلا ان قام صاحب الموقع قام بتعديل الخط في واجهة موقعه، وكتب شيئا مثل:

<p>تركبي العسيري</p>

فلو نفذت نفس البرنامج السابق لتجريد الاسم، فستكون النتيجة:

تركبي العسيري

واعتقد ان الاسم السابق ستصعب قراءته فما بالك بنطقه! اما لو لم يكن التعديل طفيف كما في الحالة بالصفحة بالمقابلة:

عملية تجريد الاسم تركي العسيري ستتطلب كتابة شيفرة برمجية لمستعرض Browser كامل!!!

أما مع لغة وصف البيانات XML، فإننا نتحدث عن عملية وصف بصيغة قياسية، فلو كان لدينا الجدول التالي:

يمكننا وصفه نصيا ونقول:

جدول الطلاب

المعرف: 1	الاسم: عباس السريع	العمر: 99
المعرف: 2	الاسم: برعي ابو جبهة	العمر: 88
المعرف: 3	الاسم: شرشيبيل بن جوزة	العمر: 77

الوصف السابق كان باللغة العربية، وهي لغة لا يتقنها سوى نسبة قليلة من البشر، لذلك سنقوم بوصفه بلغة XML وهي صيغة عالمية متوافق معها جميع التطبيقات التي تستخدمها، ونكتب شيئاً مثل:

```
<?xml version="1.0" encoding="utf-8" ?>
<table>
  <record>
    <ID>1</ID>
    <name>عباس السريع</name>
    <age>99</age>
  </record>
  <record>
    <ID>2</ID>
    <name>برعي ابو جبهة</name>
    <age>88</age>
  </record>
  <record>
    <ID>3</ID>
    <name>شرشيبيل بن جوزة</name>
    <age>77</age>
  </record>
</table>
```

بالنسبة للسطر الاول من الشيفرة السابقة، فهو يمثل توقيع ملف XML، وكما تفعل مع صفحات HTML والتي توقعها بالوسم <HTML> في اعلى الصفحة. الكلمة version هو رقم اصدار لغة XML، اما encoding فتتمثل صفحة المحارف المستخدمة في الملف، يمكنك نسخ الشيفرة السابقة وحفظها في ملف نصي بالامتداد test.xml لتتمكن من استخدامه مع كافة التطبيقات والمواقع التي تعتمد لغة XML لوصف البيانات (عليك استخدام نفس صفحة المحارف لحظة حفظ الملف).

العناصر Elements

كما رأيت في الفقرة السابقة، لغة وصف البيانات XML لا تملك كلمات محجوزة خاصة بها، وكل شيء يأتي من عندك، فهي مجرد نسق معين نتجهجه لوصف البيانات، الوسوم <xxx> التي استخدمناها تسمى **العناصر XML Elements**.

على عكس وسوم HTML، فإن عناصر XML حساسة لحالة الأحرف case-sensitive، فهي تفرق بين الحروف الكبيرة Capital والصغيرة Small. المزيد أيضا، عند انشاء عنصر من عناصر XML عليك اغلاقه دائما بكتابة نفس اسم العنصر مسبقا بالرمز /:

```
<test>
...
</test>
```

مع ذلك، ان كان العنصر لا يحتوي على قيمة، فيمكنك اغلاقه بهذه الطريقة:

```
<test />
```

اخيرا، لا تنسى ان العناصر يمكن ان تكون متداخلة:

```
<aa>
  <bb>
    <cc />
  </bb>
</aa>
```

المواصفات Attributes

المواصفات **Attributes** هي خصائص وقيم إضافية تسندها داخل العناصر، فالعنصر التالي يحتوي على مواصفتين هما X و Y، قيمة الاولى 10 والثانية 20 (يمكن ان تكون حروف ايضا):

```
<myelement x="10" y="20">value</myelemnt>
```

يمكنك الاستفادة من المواصفات في وصف أكثر للعنصر، فمثلا يمكننا إضافة المواصفة PrimaryKey لعناصر الجدول السابق لنبين بذلك ان العنصر الحالي هو مفتاح ابتدائي:

```
<?xml version="1.0" encoding="utf-8" ?>
<table>
  <record>
    <ID Primarykey="True">1</ID>
    <name>عباس السريع</name>
    <age>99</age>
  </record>
  <record>
    <ID Primarykey="True">2</ID>
    < name >برعي ابو جبهة</name>
    <age>88</age>
  </record>
  <record>
    <ID Primarykey="True">3</ID>
    <name>شرشبيب بن جوزة</name>
    <age>77</age>
  </record>
</table>
```

التعليقات Comments

اما التعليقات Comments فهي معلومات إضافية تكتب في ملف XML غرضها شرح العناصر ولا تؤثر بأي شكل من الأشكال، يمكنك كتابتها داخل التركيب <!--> و <!-->:

```
<test>
  <!--
    هذا تعليق لا يؤثر بشئ
  -->
</test>
```

خاتمة

في هذا المحلق عرضت لك لغة وصف البيانات XML والتي عبارة عن لا شيء سوى مجموعة من التنسيقات تستخدمها لوصف البيانات، نذكر أن هذه التنسيقات هي صيغ قياسية للغة XML. توجد بعض التنسيقات الإضافية يمكنك البحث عنها في مواقع الانترنت.

لغة الاستعلام SQL

إذا اردت ان تخاطب قاعدة بيانات، فانك لن تتحدث معها باللغة الصينية وانما بلغة موحدة خاصة بقواعد البيانات هي لغة الاستعلام المركبة (SQL) Structured Query Language. وهي لغة قياسية مدعومة في جميع الهيئات المختلفة لقواعد البيانات. عشرات الكتب المنتشرة هنا وهناك مختصة في لغة SQL ومئات المواقع بما فيها مكتبة MSDN- توفر لك مراجع شاملة للغة SQL، ومن الصعب علي شرح جميع اوامر وعبارات لغة SQL، لذلك يعرض لك هذا الملحق مقرر سريع للغة الاستعلام SQL تخاطب به أولئك المبرمجين الذين لم يحصل لهم شرف ممارسة واستخدام لغة SQL.

تصنيف أوامر لغة الاستعلام

بإمكاننا تصنيف أوامر وعبارات لغة SQL الى صنفين، الاول عبارات لغة تعريف البيانات **Data Definition Language (DDL)** والثاني هي عبارات لغة صيانة البيانات **Data Manipulation Language (DML)**. اوامر DDL هي أوامر وعبارات خاصة ببنية وتركيب قاعدة البيانات، فهي تمكنك من إنشاء الجداول Tables وتعرف الحقول Fields وغيرها، اما عبارات DML فهي اقرب الى الاستعلام عن البيانات في السجلات Records وإضافة وحذف سجلات اخرى، والفقرات التالية تختص ببعض اوامر DML فقط، فهي أكثر استخداما من اوامر DDL والتي يفضل المبرمجون استخدام نظام قاعدة البيانات لتعريف بنية وتركيب القاعدة عوضا عن اوامر وعبارات DDL.

الامر SELECT

يعتبر الامر SELECT بلا شك هو أكثر اوامر SQL استخداما والذي يعود بمجموعة من السجلات تحددتها في نفس الامر. المثال التالي يعود بجميع السجلات الموجودة في الجدول "بيانات الموظفين" مع جميع الحقول التابعة له:

```
SELECT * FROM [بيانات الموظفين]
```

بإمكانك تحديد حقول معينة لزيادة سرعة الاستعلام، فالمثال التالي يعود بجميع السجلات الموجودة في نفس الجدول مع تحديد حقل الاسم وتاريخ الميلاد فقط:

```
SELECT [بيانات الموظفين] FROM [تاريخ الميلاد] , [الاسم]
```

وإذا أردت استخلاص سجلات معينة توافق شرط معين استخدام العبارة WHERE، فالمثال التالي سيعود بجميع السجلات التي تكون فيها جنسية الموظف "سعودي":

```
SELECT * FROM [بيانات الموظفين]
WHERE
    [الجنسية] = 'سعودي'
```

بإمكانك استخدام ادوات الربط AND، OR وادوات المقارنة <، >، LIKE... الخ:

```
SELECT * FROM [بيانات الموظفين]
WHERE
    [الجنسية] = 'سعودي'
    AND [الاسم] LIKE 'ت%'
    OR [الراتب] < 9000
```

وإذا أردت تحديد مجال قيم معين فاستخدم المعامل BETWEEN:

```
SELECT * FROM [بيانات الموظفين]
WHERE
    [الراتب] BETWEEN 2000 AND 10000
```

او مجموعة قيم باستخدام المعامل IN:

```
SELECT * FROM [بيانات الموظفين]
WHERE
    [الجنسية] IN ('مصري', 'سعودي', 'عراقي')
```

المزيد ايضا، تستطيع فرز (ترتيب) السجلات بشكل تصاعدي باستخدام العبارة ORDER

:BY

```
SELECT * FROM [بيانات الموظفين] ORDER BY [الاسم]
```

او تنازلي باستخدام الكلمة المحجوزة DESC:

```
SELECT * FROM [بيانات الموظفين] ORDER BY [الاسم] DESC
```

الامر INSERT INTO

يمكنك الامر INSERT INTO من اضافة سجلات جديدة الى الجدول المحدد:

```
INSERT INTO [بيانات الموظفين]  
  ( [الجنسية] , [الاسم] )  
VALUES  
  ( 'سعودي' , 'تركي العسيري' )
```

الامر UPDATE

تستخدم الامر UPDATE لتحرير قيمة حقل في سجل معين تحدده في العبارة WHERE،
فالجمله التالية ستقوم بتعيين القيمة 10000 في حقل الراتب التابع للسجل الذي معرفه 32421:

```
UPDATE [بيانات الموظفين] SET  
  [الراتب] = 10000  
WHERE  
  [المعرف] = 123
```

ضع في عين الاعتبار ان التعديل قد يشمل مجموعة سجلات توافق الشرط الموجود في العبارة
WHERE، فالجمله التالية ستقوم بزيادة جميع رواتب الجنسية "سعودي" الى الضعف:

```
UPDATE [بيانات الموظفين] SET  
  [الراتب] = [الراتب] * 2  
WHERE  
  [الجنسية] = 'سعودي'
```

وان لم تكتب شرط باستخدام العبارة WHERE، فان جميع السجلات سيتم تعديلها:

```
UPDATE [بيانات الموظفين] SET  
  [الراتب] = 0
```

الامر DELETE

من الواضح ان الامر DELETE لا يقوم بعملية نسخ للسجلات وانما حذفها:

```
DELETE FROM [بيانات الموظفين]
```

في العادة لن تحذف الا عدد معين من السجلات الذي يوافق جملة شرطية باستخدام WHERE:

```
DELETE FROM [بيانات الموظفين]  
WHERE  
18 < [العمر]
```

خاتمة

في هذا الملحق عرضت مقرر سريع للغة الاستعلام SQL وهي ذات معايير ومواصفات موحدة عالميا ANSI-92 مدعومة في اغلب نظم ادارة قواعد البيانات ومزودات .NET Data Providers والتي تستخدم مع كائنات ADO.NET. تذكر أنني تحدثت عن DML وتجاهلت اوامر DDF والتي يمكنك الاستغناء عنها واستخدام نظام إدارة قواعد البيانات لإنشاء الجداول وتعريف الحقول.

